PEXlib Specification and C Language Binding

Version 5.2 - Draft Consortium Standard - rev 3

January 10, 1996

Table of Contents

Acknowledgements

On behalf of the X Consortium, we gratefully acknowledge and give thanks to the many who have participated in the design, definition and implementation of PEXlib. We acknowledge the contribution of those involved in the design and definition of the PEX protocol, without which, PEXlib would not exist.

PEXlib 5.1:

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 5.1 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 PEXlib Utilities.

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.

Last, but certainly not least, we acknowledge the first PEXlib 5.1 Chief Architect and Document editor, Jeff Stevenson (Hewlett-Packard Co.), for producing the initial and final drafts of the 5.1 specification.

PEXlib 5.2:

Thanks go to the Hewlett-Packard Co, for their contribution of the window, visual, and colormap utilities.

Thanks to HP, IBM, and SunSoft for contributing the Cose PEX extensions, from which we were able to learn from and borrow material on texture mapping.

We also thank Jeff Stevenson, Jane Sczechowski, Louis Khouw, and several other individuals and companies for helping to sweat through the design of various new interfaces for output command processing.

The Document Editors would like to acknowledge the Hewlett-Packard Company and the International Business Machines Corporation for providing time and resource needed to produce this specification.

Karl Schultz, Hewlett-Packard Company - PEXlib 5.2 Chief Architect and Document Editor.

Jennifer Kaufman, International Business Machines Corporation - PEXlib 5.2 Document Editor

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. This particular definition of PEXlib is intended for use in the X window system environment.

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., the **XSync** and **XSynchronize** functions) has 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 the **XNextEvent** function).

To query the existence of the PEX server extension, invoke the **XQueryExtension** function using the name, **PEX_NAME_STRING**, defined in PEXlib include files.

There are several instances of PEX implementations in the industry at the time of this specification that do not utilize the X protocol extension mechanism. These implementations tend to bypass the X server completely where the X server is controlling a "local" screen and direct PEX output directly to a region on the screen. These implementations are often referred to as "direct-access" implementations. Because direct-access implementations do not follow the client-sever model, the definition of a "PEX server" becomes fuzzy. In a traditional client-server implementation, the PEX extension is implemented as part of the X server and the PEX extension itself is often called a "PEX server". In the direct-access model, the function of the PEX extension is often implemented in the PEXlib library, without any particular PEX implementation in the X server itself. To accommodate both of these situations, this specification uses the word "implementation" when referring to either the PEX server extension in the traditional client-server model or the PEXlib library in the direct-access model. In either case, the implementation should behave in the same manner to the programmer and end-user. Vendors who deviate in this respect should document such differences in their product documentation.

1.1. Initializing PEXlib

Initialize PEXlib by invoking the **PEXInitialize** function before invoking any PEXlib function which takes a *display* argument either explicitly, or implicitly, via the OC Context[1]. The application must initialize PEX for each display connection that it plans to use; otherwise, results are undefined. Prior to **PEXInitialize**, the application only can invoke functions which do *not* take a *display* argument, either directly, or implicitly via the OC Context. Termination of PEXlib occurs implicitly at either program end or **XCloseDisplay**.

There are four other functions to call during initialization to determine what features and arguments are supported by each PEX server implementation. These functions are:

PEXGetImpDepConstants - returns information about supported implementation-dependent constants (page 111).
PEXGetEnumTypeInfo - returns information about supported enumerated types (page 99).
PEXMatchRenderingTargets - returns information about supported drawable types (page 120).
PEXGetExtensionInfo - returns information about the PEX server extension such as the protocol version, vendor string, release number and subset support (page 110).

A client and a server extension can use different protocol levels (for example, a PEXlib 5.2 client might connect to a PEX server extension other than a PEX 5.2 server extension). However, different major revisions of the PEX protocol

1. The OC Context is a new parameter passing mechanism introduced with PEXlib 5.2 and is described in greater detail later in this book (see "PEXlib Output Command Context (OCC) Interface" on page 134). In this discussion, it is sufficient to understand the OC Context as simply an alternate syntax for passing parameters to PEXlib.

are not compatible with each other. If the *major_version* of the PEX extension's protocol is different than 5, then PEXlib 5.2 (or any version 5 of PEXlib) won't work with the PEX extension. A PEXlib 5.2 client works with a PEX extension that has a *minor_version* level of 2 or greater. If the *minor_version* level is 0 or 1, then only a subset of the function described in this PEXlib 5.2 specification is available. The program needs to check the *minor_version* field in the data structure returned by **PEXInitialize** or **PEXGetExtensionInfo** and avoid using functions that became part of PEXlib at minor versions greater than the value in the *minor_version* field. To assist the programmer in determining which PEX features are available for each minor version of the PEX protocol, this document marks each area of functionality with the protocol level in which it was introduced. Thus, if a feature is marked(5.1), then the feature is supported only with implementations that support protocol levels of PEX 5.1 or later. An implementation that supports PEX 5.0 (returns 0 in the *minor_version* field) does not support this function. If a feature is not marked, it is assumed to be supported in all levels of PEX.

Another function, the **PEXGetSizeOCs** utility function, determines 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**), then the server issues a *BadLength* error.

1.2. PEXlib Error Handling

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 return an error event to the client. These errors are asynchronous, and so are difficult to trace to a particular Xlib or PEXlib function unless your application uses the **XSync** or **XSynchronize** function. However, use these functions prudently because they slow client programs significantly.

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

As with X11 requests, the explicit listing of **Alloc** errors only covers allocation at a very coarse level and is not intended to cover all cases of a server running out of allocation space during the processing of a request.

1.3. PEXlib Requests and Output Commands

Like the X protocol, the PEX protocol is composed of a set of requests, replies, errors, and events. About half of the functions defined in PEXlib directly generate requests, and about half do not. The latter half are the Output Command functions. Output Commands are "graphical elements" that are grouped into primitives, attributes, and control OC's. PEX defines two requests for transferring OC's to the implementation, one for storing OC's in PEX structures and one for immediate rendering. PEXlib does not define functions for these two specific requests. Instead, each OC has a corresponding PEXlib function that packs the OC into either a **PEXRCStoreElements** or **PEXRCRenderOutputCommands** request. The PEXlib implementation usually packs OC's from successive OC function invocations into a single request and keeps packing OC's into this request for as long as possible. Packing stops when the maximum request size is reached or when the application invokes a function that requires the generation of a different request. PEXlib then sends the **PEXRCStoreElements** or **PEXRCRenderOutputCommands** request to the implementation.

For more information, please see "Rendering and Storing Output Commands" on page 133.

1.3.1. Large Requests

The standard X protocol mechanism limits the maximum protocol request size to 64K words. There are several places, especially with some of the new 5.2 functions, where you may exceed this limit. The texture map creation and Put Z Buffer functions can easily exceed this limit. Large output commands, discussed below, can also exceed the limit. In response to this problem, the X Consortium has specified the BRE (Big Request Extension). X servers that support PEX 5.2 should also support BRE to allow the PEX client to use large output commands and to work with large textures and Z buffer data. If the X server does not support the BRE, then PEXlib 5.2 clients need to take precautions to avoid generating requests that are too large when working with these larger data items.

1.3.2. Large Output Commands

Prior to PEX 5.2, the largest output command was limited to 64K words in size. PEX 5.2 now employs a "large OC" mechanism that is very similar to the approach that the BRE uses. Of course, in order to store or render large OC's, the BRE must be present to correctly process the larger request required to transport the large OC's to the server. Again, if the BRE is not present, the PEXlib application must take care to avoid generating OC's that are larger than 64K words.

1.4. PEX Resources

PEX defines a number of resources that you create, manipulate, and destroy, using the functions defined in this specification. These resources behave identically to X resources. For example, PEX resources may be shared among clients and are automatically destroyed by the implementation when no longer referenced.

Some PEX resources contain references to other PEX resources. For example, the Pipeline Context resource contains references to PEX Name Set resources. If you reference a resource from another resource and then destroy the referenced resource, then you can no longer access the destroyed resource using PEXlib functions because the resource identifier is no longer valid. However, the destroyed resource actually remains intact while it is still being referenced by the other resource. This is necessary because the implementation may still need to access the resource during a rendering operation, for example. If a resource is destroyed, then the implementation changes the resource identifiers in all references to the destroyed resource to the value **PEXAlreadyFreed** to indicate that this resource has been destroyed and cannot be manipulated further. You can replace the reference to the resource id of a freed resource with that of another existing resource. When the last reference to a freed resource is removed, the implementation actually frees the resource.

In some instances, you may want to replace a resource reference with a null reference or something that indicates that no resource at all is referenced. For example, you may be using a depth cue LUT resource by referencing it from a renderer resource. If you want the renderer to go back to using the default PEX-defined fallback values for depth cue tables, you remove the renderer's reference to the depth cue LUT resource by changing the renderer's depth cue LUT resource attribute to **None**. **None** is defined in **<X11/Xlib.h>**.

1.5. Language Bindings Supported

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.

1.6. The PEXlib Include Files

The PEXlib specification currently defines include files for the C language only. The following include files contain constant definitions, macros, and data structures used by PEXlib: **<X11/PEX5/PEX.h>** and **<X11/PEX5/PEXlib.h>**. **<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. PEXlib ignores any values assigned to these fields.

1.7. Compiling and Linking PEXlib Programs

You should include the PEXlib header files with the following C language preprocessor directive inserted in your program:

```
#include <X11/PEX5/PEXlib.h>
```

In most cases, this is sufficient for the compiler to find the file. On some systems, you may need to specify the "`-I`" option on your C compiler's command line to locate the files.

When linking, specify the library with the option: `-lPEX5`. On some systems, you may need to use the “`-L`” option or specify the full path name of the library to help the linker locate the file.

The utility functions in *Chapter 25* use math functions which create a dependency on the math library. This means that all PEXlib applications need to link to the math library, typically specified in the compilation command as `-lm`.

1.8. PEXlib Floating-Point Format

PEXlib uses the floating-point format native to the architecture of the client machine (i.e., the machine on which the application executes). PEXlib expects all of the floating-point values to be in that native format, with a few exceptions for functions which allow the application to specify the floating-point format. These functions, which enable 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.

If the server to which PEXlib is sending data does not support the floating-point format native to PEXlib, then PEXlib chooses an appropriate format which is supported by the server and converts data 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 invoking the **PEXGetProtocolFloatFormat** function. This is useful with functions dealing with protocol formatted data. It helps avoid unnecessary format conversions when sending the data to the PEX server.

1.9. Freeing Memory Allocated by PEXlib functions

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 the **XFree** function to deallocate memory that has been allocated for them by PEXlib, unless PEXlib provides a specific function to free memory allocated for particular PEXlib objects. Where necessary, the specific function for deallocation is noted in the description of each function that allocates the memory. Memory allocated by PEXlib is considered to be ‘owned’ by PEXlib; the application must not modify pointers to memory that the library returns nor attempt to free such memory except by the specified functions. The application can change data structure fields other than pointers to memory without consequences.

Table 1: Functions for Freeing Memory Allocated by PEXlib

Functions that Allocate Memory	Functions to Deallocate Memory
PEXCreateOCC (page 148)	PEXFreeOCC (page 149)
PEXConvertDoubleFloat (page 86)	XFree
PEXConvertSingleFloat (page 88)	XFree
PEXDecodeOCs (page 728)	PEXFreeOCData (page 730)
PEXEncodeOCs (page 729)	XFree
PEXEndPickAll (page 567)	PEXFreePickPaths (page 570)
PEXEndPickOne (page 569)	PEXFreePickPaths (page 570)
PEXEscapeWithReply (page 723)	XFree
PEXFetchElements (page 591)	XFree
PEXGenerateMipmap (page 628)	PEXFreeMipmap (page 627)

Table 1: Functions for Freeing Memory Allocated by PEXlib

Functions that Allocate Memory	Functions to Deallocate Memory
PEXGenerateMipmapFromResources (page 630)	PEXFreeMipmap (page 627)
PEXGetAncestors (page 596)	PEXFreeStructurePaths (page 595)
PEXGetAncillaryBufferInfo (page 97)	PEXFreeAncillaryBufferInfo (page 95)
PEXGetDescendants (page 598)	PEXFreeStructurePaths (page 595)
PEXGetDefinedIndices (page 480)	XFree
PEXGetElementInfo (page 600)	XFree
PEXGetEnumTypeInfo (page 99)	PEXFreeEnumInfo (page 96)
PEXGetImpDepConstants (page 111)	XFree
PEXGetLargeElementInfo (page 600)	XFree
PEXGetNameSet (page 616)	XFree
PEXGetPickDevice (page 706)	PEXFreePDAttributes (page 703)
PEXGetPickMeasure (page 707)	PEXFreePMAttributes (page 705)
PEXGetPredefinedEntries (page 485)	PEXFreeTableEntries (page 484)
PEXGetRendererExtAttributes (page 537)	PEXFreeRendererExtAttributes (page 536)
PEXGetRendererAttributes (page 556)	PEXFreeRendererAttributes (page 555)
PEXGetSearchContext (page 666)	PEXFreeSCAttributes (page 664)
PEXGetStructuresInNetwork (page 605)	XFree
PEXGetTableEntry (page 489)	PEXFreeTableEntries (page 484)
PEXGetTableEntries (page 487)	PEXFreeTableEntries (page 484)
PEXGetTableValues (page 482)	PEXFreeTableValues (page 479)
PEXGetWorkstationAttributes (page 678)	PEXFreeWorkstationAttributes (page 676)
PEXGetWorkstationPostings (page 679)	XFree
PEXGetZBuffer (page 538)	XFree
PEXListFontsWithInfo (page 713)	PEXFreeFontNames (page 711), PEXFreeFontInfo (page 710)
PEXListFonts (page 712)	PEXFreeFontNames (page 711)
PEXMapDCToWC (page 681)	XFree
PEXMapWCToDC (page 682)	XFree
PEXMatchRenderingTargets (page 120)	XFree
PEXPickAll (page 571)	PEXFreePickPaths (page 570)
PEXPickOne (page 565)	PEXFreePickPaths (page 570)

Table 1: Functions for Freeing Memory Allocated by PEXlib

Functions that Allocate Memory	Functions to Deallocate Memory
PEXQueryEncodedTextExtents	XFree
PEXQueryFont (page 717)	PEXFreeFontInfo (page 710)
PEXQueryTextExtents (page 718)	XFree
PEXSearchNetwork (page 667)	PEXFreeStructurePaths (page 595)

1.10. PEXlib Conventions

The following conventions apply to PEXlib:

- PEXlib uses mixed case for external symbols. PEXlib leaves all lowercase 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 (back-facing), HLHSR (hidden-line/hidden surface removal), LUT (lookup table), OC (output command), PC (pipeline context).
- All PEXlib functions, symbols, macros, user-visible data structures or anything a user might reference begin with "PEX". A single exception to this is that the error constants have a prefix of "BadPEX" for consistency 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 lowercase. 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 the first argument. Often, the OC Context parameter *context* is the first parameter and contains the *display* argument.
- All resource arguments, where needed, occur at the beginning of the argument list but after the *display* or *context* argument. Where a drawable resource is needed, it is 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.
- Color quads are listed in r,g,b,a order.
- 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.

1.11. Vendor-Specific Extensions and Conventions

Some vendors may choose to extend the PEX protocol with a set of specific functionality. To access the extension from your program easily, the vendor provides a file containing constant definitions, macros, and data structures that pertain to the extension functionality. These include files are found in **<X11/PEX5/extensions>** and abide by the following conventions:

- provides one single file for a "logical group" of extensions
- capitalizes vendor names, "ACME" in the following examples.
- makes sincere effort to use existing PEXlib conventions (listed above) where possible
- uses the filename format: PEX*VENDORExtensionName*.h (e.g., `PEXACMEDraftingOCs.h`)
- uses the constant name format: PEX*VENDORConstantName* (e.g., `PEXACMEArcType`)

- uses the typedef name format: PEX*VENDORTypedefName* (e.g., `PEXACMEDatatype`)
- uses the error name format: BadPEX*VENDORErrorName* (e.g., `BadPEXACMEIdea`)
- assumes **PEXlib.h** has already been included
- offers re-include protection in the files: `#ifndef _FILENAME_H_`, e.g.:

```
#ifndef _PEXACMEDRAFTINGOCS_H_
```

1.11.1. Vendor-Specific Escapes, GDP's, and GSE's

Vendors that implement their own specific escapes, generalized drawing primitives, or generalized structure elements should supply a C header file that contains at least three definitions:

- The 32-bit ID of the Escape, GDP,or GSE - example: `#define PEXACMEEscapeBlink 0x80010005`
- The 16-bit enumerated type value - example: `#define PEXACMEETEscapeBlink 0x8105`
- The enumerated type mnemonic - example `#define PEXACMEETMEscapeBlink "ACMEEscapeBlink"`

The vendor should also provide data structure definitions, if the client program needs them to format the data it supplies for these extensions.

It is good practice for clients to use both the enumerated type value and the enumerated type mnemonic when searching for matches in the output returned from **PEXGetEnumTypeInfo**. This is because it is possible for more than one 32-bit ID to map to the same 16-bit enumerated type value. Use the mnemonic to distinguish between like values.

Chapter 2

PEXlib Data Structures

2.1. Grouped by Categories

The PEXlib data structures are grouped in the following categories.

2.2. Data Structure Files

2.2.1. PEX.h

This file defines names for constant values that you need when building a PEXlib application. These constants mainly represent PEX protocol encoding values that you do not need to know by value, but need to know symbolically instead. The names of these constants are used throughout the specification, but since their values are not usually important to understanding or using PEXlib, the constants are not referred to by value in this document.

2.2.2. PEXlib.h

This file contains data structures (type definitions) and function prototypes. The data structures are repeated here for your programming convenience, with the exception of `PEXOCData`, which is too large to reproduce here.

2.3. Commonly Used Type Definitions

The data structures for type definitions include

2.3.1. `PEXArcAreaType` Type Definition

```
typedef unsigned int        PEXArcAreaType;
```

2.3.2. `PEXArcType` Type Definition

```
typedef unsigned int        PEXArcType;
```

2.3.3. `PEXBitmask` Type Definition

```
typedef unsigned long       PEXBitmask;
```

2.3.4. `PEXColorType` Type Definition

```
typedef short               PEXColorType;
```

2.3.5. `PEXCoordType` Type Definition

```
typedef unsigned short      PEXCoordType;
```

2.3.6. `PEXEnumTypeIndex` Type Definition

```
typedef short               PEXEnumTypeIndex;
```

2.3.7. `PEXFont` Type Definition

```
typedef XID                 PEXFont;
```

2.3.8. `PEXLookupTable` Type Definition

```
typedef XID                 PEXLookupTable;
```

2.3.9. `PEXMatrix3x3` Type Definition

```
typedef float               PEXMatrix3x3[3][3];
```

2.3.10. `PEXMatrix` Type Definition

```
typedef float               PEXMatrix[4][4];
```

2.3.11. `PEXMatrixHints` Type Definition

```
typedef unsigned long       PEXMatrixHints;
```

2.3.12. `PEXName` Type Definition

```
typedef unsigned long       PEXName;
```

2.3.13. `PEXNameSet` Type Definition

```
typedef XID                 PEXNameSet;
```

2.3.14. `PEXOCRequestType` Type Definition

```
typedef int                 PEXOCRequestType;
```

2.3.15. `PEXPickMeasure` Type Definition

```
typedef XID                 PEXPickMeasure;
```

2.3.16. `PEXPipelineContext` Type Definition

```
typedef XID                 PEXPipelineContext;
```

2.3.17. `PEXRenderer` Type Definition

```
typedef XID                   PEXRenderer;
```

2.3.18. `PEXSearchContext` Type Definition

```
typedef XID                   PEXSearchContext;
```

2.3.19. `PEXStructure` Type Definition

```
typedef XID                   PEXStructure;
```

2.3.20. `PEXSwitch` Type Definition

```
typedef unsigned char         PEXSwitch;
```

2.3.21. `PEXTableIndex` Type Definition

```
typedef unsigned short        PEXTableIndex;
```

2.3.22. `PEXTextureMap` Type Definition

```
typedef     XID               PEXTextureMap;
```

2.3.23. `PEXTypeOrTableIndex` Type Definition

```
typedef unsigned short        PEXTypeOrTableIndex;
```

2.3.24. `PEXWorkstation` Type Definition

```
typedef XID                   PEXWorkstation;
```

2.4. Data Structures for Ancillary Buffer Information

The data structures for ancillary buffer information include:

2.4.1. `PEXAncillaryBufferInfo` Data Structure

```
typedef struct {
      unsigned short z_depth;
      unsigned short z_format;
      union {
            struct {
                  unsigned long z_min;
                  unsigned long z_max;
            } integer;
            struct {
                  float z_min;
                  float z_max;
            } flt;
      } z_minmax;
      unsigned short alpha_depth;
      unsigned short alpha_format;
      union {
            struct {
                  unsigned long alpha_min;
                  unsigned long alpha_max;
            } integer;
            struct {
```

```
                        float alpha_min;
                        float alpha_max;
                  } flt;
            } alpha_minmax;
            unsigned short accum_depth;
            unsigned short accum_depth_red;
            unsigned short accum_depth_green;
            unsigned short accum_depth_blue;
            unsigned short accum_depth_alpha;
      } PEXAncillaryBufferInfo;
```

2.5. Data Structures for Colors

The data structures for colors include:

2.5.1. `PEXColorIndexed` Data Structure

```
      typedef struct {
            PEXTableIndex     index; [§2.3.21, page 29]
            unsigned short    reserved;
      } PEXColorIndexed;
```

2.5.2. `PEXColorCIE` Data Structure

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

2.5.3. `PEXColorHLS` Data Structure

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

2.5.4. `PEXColorHSV` Data Structure

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

2.5.5. `PEXColorRGB` Data Structure

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

2.5.6. `PEXColorRGBA` Data Structure

```
typedef struct {
      float             red;
      float             green;
      float             blue;
      float             alpha;
} PEXColorRGBA;
```

2.5.7. `PEXColorRGB8` Data Structure

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

2.5.8. `PEXColorRGBA8` Data Structure

```
typedef struct {
      unsigned char     red;
      unsigned char     green;
      unsigned char     blue;
      unsigned char     alpha;
} PEXColorRGBA8;
```

2.5.9. `PEXColorRGB16` Data Structure

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

2.5.10. `PEXColorRGBA16` Data Structure

```
typedef struct {
      unsigned short    red;
      unsigned short    green;
      unsigned short    blue;
      unsigned short    alpha;
} PEXColorRGBA16;
```

2.5.11. `PEXArrayOfColor` Data Structure

```
typedef union {
      PEXColorIndexed   *indexed; [§2.5.1, page 30]
      PEXColorCIE       *cie; [§2.5.2, page 30]
      PEXColorHLS       *hls; [§2.5.3, page 30]
      PEXColorHSV       *hsv; [§2.5.4, page 30]
      PEXColorRGB       *rgb; [§2.5.5, page 30]
      PEXColorRGBA      *rgba; [§2.5.6, page 31]
      PEXColorRGB8      *rgb8; [§2.5.7, page 31]
      PEXColorRGBA8     *rgba8; [§2.5.8, page 31]
      PEXColorRGB16     *rgb16; [§2.5.9, page 31]
      PEXColorRGBA16    *rgba16; [§2.5.10, page 31]
} PEXArrayOfColor;
```

2.5.12. `PEXColor` Data Structure

```
typedef union {
      PEXColorIndexed   indexed; [§2.5.1, page 30]
      PEXColorCIE       cie; [§2.5.2, page 30]
      PEXColorHLS       hls; [§2.5.3, page 30]
      PEXColorHSV       hsv; [§2.5.4, page 30]
      PEXColorRGB       rgb; [§2.5.5, page 30]
      PEXColorRGB8      rgb8; [§2.5.7, page 31]
      PEXColorRGB16     rgb16; [§2.5.9, page 31]
} PEXColor;
```

2.5.13. `PEXColor5_2` Data Structure

```
typedef union {
      PEXColorIndexed   indexed; [§2.5.1, page 30]
      PEXColorCIE       cie; [§2.5.2, page 30]
      PEXColorHLS       hls; [§2.5.3, page 30]
      PEXColorHSV       hsv; [§2.5.4, page 30]
      PEXColorRGB       rgb; [§2.5.5, page 30]
      PEXColorRGBA      rgba; [§2.5.6, page 31]
      PEXColorRGB8      rgb8; [§2.5.7, page 31]
      PEXColorRGBA8     rgba8; [§2.5.8, page 31]
      PEXColorRGB16     rgb16; [§2.5.9, page 31]
      PEXColorRGBA16    rgba16; [§2.5.10, page 31]
} PEXColor5_2;
```

2.5.14. `PEXColorSpecifier` Data Structure

```
typedef struct {
      PEXColorType      type; [§2.3.4, page 28]
      unsigned short    reserved;
      PEXColor          value; [§2.5.12, page 32]
} PEXColorSpecifier;
```

2.5.15. `PEXColorSpecifier5_2` Data Structure

```
typedef struct {
      PEXColorType      type; [§2.3.4, page 28]
      unsigned short    reserved;
      PEXColor5_2       value; [§2.5.13, page 32]
} PEXColorSpecifier5_2;
```

2.6. Data Structures for Connectivity

The data structures for connectivity include:

2.6.1. `PEXListOfUShort` Data Structure

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

2.6.2. `PEXConnectivityData` Data Structure

```
typedef struct {
      unsigned short    count;                  /* number of lists */
      PEXListOfUShort   *lists; [§2.6.1, page 32]
} PEXConnectivityData;
```

2.7. Data Structures for Coordinates

The data structures for coordinates include:

2.7.1. `PEXCoord` Data Structure

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

2.7.2. `PEXCoord2D` Data Structure

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

2.7.3. `PEXCoord4D` Data Structure

```
typedef struct {
      float             x;
      float             y;
      float             z;
      float             w;
} PEXCoord4D;
```

2.7.4. `PEXArrayOfCoord` Data Structure

```
typedef union {
      PEXCoord2D        *point_2d; [§2.7.2, page 33]
      PEXCoord          *point; [§2.7.1, page 33]
      PEXCoord4D        *point_4d; [§2.7.3, page 33]
} PEXArrayOfCoord;      /* Pointer to array of points */
```

2.7.5. `PEXListOfCoord` Data Structure

```
typedef struct {
      unsigned long     count;       /* number of points */
      PEXCoord          *points; [§2.7.1, page 33]
} PEXListOfCoord;                    /* Pointer to an array of 3D points */
```

2.7.6. `PEXListOfCoord2D` Data Structure

```
typedef struct {
      unsigned long     count;       /* number of points */
      PEXCoord2D        *points; [§2.7.2, page 33]
} PEXListOfCoord2D;                  /* Pointer to an array of 2D points */
```

2.8. Data Structures for Encoded Text

The data structures for encoded text include

2.8.1. `PEXEncodedTextData` Data Structure

```
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;
```

2.9. Data Structures for Enumerated Types

The data structures for enumerated types include:

2.9.1. `PEXEnumTypeDesc` Data Structure

```
typedef struct {
      PEXEnumTypeIndex  index; [§2.3.6, page 28]
      char              *descriptor;          /* null terminated string */
} PEXEnumTypeDesc;
```

2.10. Data Structures for Error Events

The data structures for error events include:

```
/* similar to XErrorEvent - use to access additional info in OC error */
typedef struct {
      int               type;
      Display           *display;
      XID               resourceid;
      unsigned long     serial;
      unsigned char     error_code;
      unsigned char     request_code;
      unsigned char     minor_code;
      unsigned short    op_code;
      unsigned short    count;
} PEXOCErrorEvent;
```

2.11. Data Structures for PEX Extensions

The data structures for PEX extensions include:

2.11.1. `PEXExtensionInfo` Data Structure

```
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;
```

2.12. Data Structures for Facet Data

The data structures for facet data include:

2.12.1. `PEXColorIndexedNormal` Data Structure

```
typedef struct {
      PEXColorIndexed   index; [§2.5.1, page 30]
      PEXVector         normal; [§2.31.1, page 67]
} PEXColorIndexedNormal;
```

2.12.2. `PEXColorCIENormal` Data Structure

```
typedef struct {
      PEXColorCIE       cie; [§2.5.2, page 30]
      PEXVector         normal; [§2.31.1, page 67]
} PEXColorCIENormal;
```

2.12.3. `PEXColorHLSNormal` Data Structure

```
typedef struct {
      PEXColorHLS       hls; [§2.5.3, page 30]
      PEXVector         normal; [§2.31.1, page 67]
} PEXColorHLSNormal;
```

2.12.4. `PEXColorHSVNormal` Data Structure

```
typedef struct {
      PEXColorHSV       hsv; [§2.5.4, page 30]
      PEXVector         normal; [§2.31.1, page 67]
} PEXColorHSVNormal;
```

2.12.5. `PEXColorRGBNormal` Data Structure

```
typedef struct {
      PEXColorRGB       rgb; [§2.5.5, page 30]
      PEXVector         normal; [§2.31.1, page 67]
} PEXColorRGBNormal;
```

2.12.6. `PEXColorRGBANormal` Data Structure

```
typedef struct {
      PEXColorRGBA      rgba; [§2.5.6, page 31]
      PEXVector         normal; [§2.31.1, page 67]
} PEXColorRGBANormal;
```

2.12.7. `PEXColorRGB8Normal` Data Structure

```
typedef struct {
      PEXColorRGB8      rgb8; [§2.5.7, page 31]
      PEXVector         normal; [§2.31.1, page 67]
} PEXColorRGB8Normal;
```

2.12.8. `PEXColorRGBA8Normal` Data Structure

```
typedef struct {
      PEXColorRGBA8     rgba8; [§2.5.8, page 31]
      PEXVector         normal; [§2.31.1, page 67]
} PEXColorRGBA8Normal;
```

2.12.9. `PEXColorRGB16Normal` Data Structure

```
typedef struct {
      PEXColorRGB16     rgb16; [§2.5.9, page 31]
      PEXVector         normal; [§2.31.1, page 67]
} PEXColorRGB16Normal;
```

2.12.10. `PEXColorRGBA16Normal` Data Structure

```
typedef struct {
      PEXColorRGBA16    rgba16; [§2.5.10, page 31]
      PEXVector         normal; [§2.31.1, page 67]
} PEXColorRGBA16Normal;
```

2.12.11. `PEXArrayOfFacetData` Data Structure

```
typedef union {
      PEXColorIndexed         *index; [§2.5.1, page 30]
      PEXColorCIE             *cie; [§2.5.2, page 30]
      PEXColorHLS             *hls; [§2.5.3, page 30]
      PEXColorHSV             *hsv; [§2.5.4, page 30]
      PEXColorRGB             *rgb; [§2.5.5, page 30]
      PEXColorRGBA            *rgba; [§2.5.6, page 31]
      PEXColorRGB8            *rgb8; [§2.5.7, page 31]
      PEXColorRGBA8           *rgba8; [§2.5.8, page 31]
      PEXColorRGB16           *rgb16; [§2.5.9, page 31]
      PEXColorRGBA16          *rgba16; [§2.5.10, page 31]
      PEXVector               *normal; [§2.31.1, page 67]
      PEXColorIndexedNormal   *index_normal; [§2.12.1, page 35]
      PEXColorCIENormal       *cie_normal; [§2.12.2, page 35]
      PEXColorHLSNormal       *hls_normal; [§2.12.3, page 35]
      PEXColorHSVNormal       *hsv_normal; [§2.12.4, page 35]
      PEXColorRGBNormal       *rgb_normal; [§2.12.5, page 35]
      PEXColorRGBANormal      *rgba_normal; [§2.12.6, page 35]
      PEXColorRGB8Normal      *rgb8_normal; [§2.12.7, page 36]
      PEXColorRGBA8Normal     *rgba8_normal; [§2.12.8, page 36]
      PEXColorRGB16Normal     *rgb16_normal; [§2.12.9, page 36]
      PEXColorRGBA16Normal    *rgba16_normal; [§2.12.10, page 36]
} PEXArrayOfFacetData;
```

2.12.12. PEXFacetData Data Structure

```
typedef union {
      PEXColorIndexed         index; [§2.5.1, page 30]
      PEXColorCIE             cie; [§2.5.2, page 30]
      PEXColorHLS             hls; [§2.5.3, page 30]
      PEXColorHSV             hsv; [§2.5.4, page 30]
      PEXColorRGB             rgb; [§2.5.5, page 30]
      PEXColorRGBA            rgba; [§2.5.6, page 31]
      PEXColorRGB8            rgb8; [§2.5.7, page 31]
      PEXColorRGBA8           rgba8; [§2.5.8, page 31]
      PEXColorRGB16           rgb16; [§2.5.9, page 31]
      PEXColorRGBA16          rgba16; [§2.5.10, page 31]
      PEXVector               normal; [§2.31.1, page 67]
      PEXColorIndexedNormal   index_normal; [§2.12.1, page 35]
      PEXColorCIENormal       cie_normal; [§2.12.2, page 35]
      PEXColorHLSNormal       hls_normal; [§2.12.3, page 35]
      PEXColorHSVNormal       hsv_normal; [§2.12.4, page 35]
      PEXColorRGBNormal       rgb_normal; [§2.12.5, page 35]
      PEXColorRGBANormal      rgba_normal; [§2.12.6, page 35]
      PEXColorRGB8Normal      rgb8_normal; [§2.12.7, page 36]
      PEXColorRGBA8Normal     rgba8_normal; [§2.12.8, page 36]
      PEXColorRGB16Normal     rgb16_normal; [§2.12.9, page 36]
      PEXColorRGBA16Normal    rgba16_normal; [§2.12.10, page 36]
} PEXFacetData;
```

2.13. Data Structures for Fonts

The data structures for fonts include:

2.13.1. PEXEncodedTextData Data Structure

```
typedef struct {
      unsigned short    character_set;
      unsigned char     character_set_width;
      unsigned char     encoding_state;
      unsigned short    reserved;
      unsigned short    length;
      char              *ch;
} PEXEncodedTextData;
```

2.13.2. PEXFontInfo Data Structure

```
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; [§2.13.3, page 38]
} PEXFontInfo;
```

2.13.3. PEXFontProp Data Structure

```
typedef struct {
      Atom               name;
      unsigned long      value;
} PEXFontProp;
```

2.13.4. PEXFontSet Data Structure

```
typedef struct {
      PEXTextFontEntry         font_LUT_entry; [§2.16.26, page 44]
      int                      missing_charset_count;
      char                     *locale_string_copy;
      char                     **req_name_list;
      char                     **font_name_list;
      char                     *default_string;
      PEXTextEncoding          methods; [§2.13.7, page 38]
} PEXFontSet;

#define PEXFSFontIDList               (1L<<0)
#define PEXFSLocaleString             (1L<<1)
#define PEXFSReqFontNameList          (1L<<2)
#define PEXFSFontNameList             (1L<<3)
#define PEXFSDefaultString            (1L<<4)
#define PEXFSAll                      ~0
```

2.13.5. PEXListOfEncodedText Data Structure

```
typedef struct {
      unsigned short           count;              /* number of encodings */
      PEXEncodedTextData       *encoded_text; [§2.13.1, page 37]
} PEXListOfEncodedText;
```

2.13.6. PEXStringData Data Structure

```
typedef struct {
      unsigned short     length;
      char               *ch;
} PEXStringData;
```

2.13.7. PEXTextEncoding Data Structure

```
typedef void*            PEXTextEncoding;
```

2.13.8. PEXTextExtent Data Structure

```
typedef struct {
      PEXCoord2D         lower_left; [§2.7.2, page 33]
      PEXCoord2D         upper_right; [§2.7.2, page 33]
      PEXCoord2D         concat_point; [§2.7.2, page 33]
} PEXTextExtent;
```

2.13.9. PEXWStringData Data Structure

```
typedef struct {
      unsigned short     length;
      wchar_t            *wch;
} PEXWStringData;
```

2.14. Data Structures for Half Spaces

The data structures for half spaces include:

2.14.1. `PEXHalfSpace` Data Structure

```
typedef struct {
      PEXCoord          point; [§2.7.1, page 33]
      PEXVector         vector; [§2.31.1, page 67]
} PEXHalfSpace;
```

2.14.2. `PEXHalfSpace2D` Data Structure

```
typedef struct {
      PEXCoord2D        point; [§2.7.2, page 33]
      PEXVector2D       vector; [§2.31.2, page 67]
} PEXHalfSpace2D;
```

2.15. Data Structures for Implementation-Dependent Constants

The data structures for implementation-dependent constants include:

2.15.1. `PEXImpDepConstant`Data Structure

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

2.16. Data Structures for Lookup Tables

The data structures for lookup tables include:

2.16.1. `PEXColorEntry` Type Definition

```
typedef PEXColorSpecifier     PEXColorEntry;
```

2.16.2. `PEXColorValues` Type Definition

```
typedef PEXColorEntry PEXColorValues;
```

2.16.3. `PEXColorApproxEntry` Data Structure

```
typedef struct {
```

```
        PEXEnumTypeIndex  type; [§2.3.6, page 28]
        PEXEnumTypeIndex  model; [§2.3.6, page 28]
        unsigned short    max1;
        unsigned short    max2;
        unsigned short    max3;
        PEXSwitch         dither; [§2.3.20, page 29]
        unsigned char     reserved;
        unsigned long     mult1;
        unsigned long     mult2;
        unsigned long     mult3;
        float             weight1;
        float             weight2;
        float             weight3;
        unsigned long     base_pixel;
} PEXColorApproxEntry;
```

2.16.4. `PEXColorApproxValues` Type Definition

```
typedef PEXColorApproxEntry PEXColorApproxValues;
```

2.16.5. `PEXDataMappingValues` Data Structure

```
typedef struct {
        PEXEnumTypeIndex        method; [§2.3.6, page 28]
        unsigned short          count;
        unsigned short          *source_selectors;
} PEXDataMappingValues;
```

2.16.6. `PEXDepthCueEntry` Data Structure

```
typedef struct {
        PEXSwitch         mode; [§2.3.20, page 29]
        unsigned char     reserved[3];
        float             front_plane;
        float             back_plane;
        float             front_scaling;
        float             back_scaling;
        PEXColorSpecifier color; [§2.5.14, page 32]
} PEXDepthCueEntry;
```

2.16.7. `PEXDepthCueValues` Data Structure

```
typedef struct {
        PEXSwitch         mode; [§2.3.20, page 29]
        unsigned char     reserved[3];
        float             front_plane;
        float             back_plane;
        float             front_scaling;
        float             back_scaling;
        PEXColorSpecifier5_2 color; [§2.5.15, page 32]
} PEXDepthCueValues;
```

2.16.8. `PEXEdgeBundleEntry` Data Structure

```
typedef struct {
      PEXSwitch               edge_flag; [§2.3.20, page 29]
      unsigned char           reserved;
      PEXEnumTypeIndex        type; [§2.3.6, page 28]
      float                   width;
      PEXColorSpecifier       color; [§2.5.14, page 32]
} PEXEdgeBundleEntry;
```

2.16.9. `PEXEdgeBundleValues` Data Structure

```
typedef struct {
      PEXSwitch               edge_flag; [§2.3.20, page 29]
      unsigned char           reserved;
      PEXEnumTypeIndex        type; [§2.3.6, page 28]
      float                   width;
      PEXColorSpecifier5_2    color; [§2.5.15, page 32]
} PEXEdgeBundleValues;
```

2.16.10. `PEXExtentSizeValues` Data Structure

```
typedef struct {
      unsigned long     extent_size;
} PEXExtentSizeValues;
```

2.16.11. `PEXHighlightValues` Data Structure

```
typedef struct {
      PEXEnumTypeIndex        method; [§2.3.6, page 28]
      unsigned short          reserved;
      PEXColorSpecifier5_2    color; [§2.5.15, page 32]
} PEXHighlightValues;
```

2.16.12. `PEXInteriorBundleEntry` Data Structure

```
typedef struct {
      PEXEnumTypeIndex        style; [§2.3.6, page 28]
      PEXTypeOrTableIndex     style_index; [§2.3.23, page 29]
      PEXEnumTypeIndex        reflection_model; [§2.3.6, page 28]
      PEXEnumTypeIndex        interp_method; [§2.3.6, page 28]
      PEXEnumTypeIndex        bf_style; [§2.3.6, page 28]
      PEXTypeOrTableIndex     bf_style_index; [§2.3.23, page 29]
      PEXEnumTypeIndex        bf_reflection_model; [§2.3.6, page 28]
      PEXEnumTypeIndex        bf_interp_method; [§2.3.6, page 28]
      PEXSurfaceApprox        surface_approx; [§2.24.12, page 59]
      PEXColorSpecifier       color; [§2.5.14, page 32]
      PEXReflectionAttributes reflection_attr; [§2.24.10, page 58]
      PEXColorSpecifier       bf_color; [§2.5.14, page 32]
      PEXReflectionAttributes bf_reflection_attr; [§2.24.10, page 58]
} PEXInteriorBundleEntry;
```

2.16.13. PEXInteriorBundleValues Data Structure

```
typedef struct {
      PEXEnumTypeIndex        style; [§2.3.6, page 28]
      PEXTypeOrTableIndex     style_index; [§2.3.23, page 29]
      PEXEnumTypeIndex        reflection_model; [§2.3.6, page 28]
      PEXEnumTypeIndex        interp_method; [§2.3.6, page 28]
      PEXEnumTypeIndex        bf_style; [§2.3.6, page 28]
      PEXTypeOrTableIndex     bf_style_index; [§2.3.23, page 29]
      PEXEnumTypeIndex        bf_reflection_model; [§2.3.6, page 28]
      PEXEnumTypeIndex        bf_interp_method; [§2.3.6, page 28]
      PEXSurfaceApprox        surface_approx; [§2.24.12, page 59]
      PEXColorSpecifier5_2    color; [§2.5.15, page 32]
      PEXReflectionAttributes5_2 reflection_attr; [§2.24.11, page 59]
      PEXColorSpecifier5_2    bf_color; [§2.5.15, page 32]
      PEXReflectionAttributes5_2 bf_reflection_attr; [§2.24.11, page 59]
      PEXPSC                  para_surf; [§2.22.1, page 51]
} PEXInteriorBundleValues;
```

2.16.14. PEXLightEntry Data Structure

```
typedef struct {
      PEXEnumTypeIndex  type; [§2.3.6, page 28]
      unsigned short    reserved;
      PEXVector         direction; [§2.31.1, page 67]
      PEXCoord          point; [§2.7.1, page 33]
      float             concentration;
      float             spread_angle;
      float             attenuation1;
      float             attenuation2;
      PEXColorSpecifier color; [§2.5.14, page 32]
} PEXLightEntry;
```

2.16.15. PEXLightValues Data Structure

```
typedef struct {
      PEXEnumTypeIndex  type; [§2.3.6, page 28]
      unsigned short    reserved;
      PEXVector         direction; [§2.31.1, page 67]
      PEXCoord          point; [§2.7.1, page 33]
      float             concentration;
      float             spread_angle;
      float             attenuation1;
      float             attenuation2;
      PEXColorSpecifier5_2 color; [§2.5.15, page 32]
      float             concentration_clamp_region;
      float             quadratic_atten_factor;
} PEXLightValues;
```

2.16.16. PEXLineBundleEntry Data Structure

```
typedef struct {
      PEXEnumTypeIndex  type; [§2.3.6, page 28]
      PEXEnumTypeIndex  interp_method; [§2.3.6, page 28]
      PEXCurveApprox    curve_approx; [§2.24.3, page 58]
      float             width;
      PEXColorSpecifier color; [§2.5.14, page 32]
} PEXLineBundleEntry;
```

2.16.17. `PEXLineBundleValues` Data Structure

```
typedef struct {
      PEXEnumTypeIndex  type; [§2.3.6, page 28]
      PEXEnumTypeIndex  interp_method; [§2.3.6, page 28]
      PEXCurveApprox    curve_approx; [§2.24.3, page 58]
      float             width;
      PEXColorSpecifier5_2 color; [§2.5.15, page 32]
      PEXEnumTypeIndex  cap_style; [§2.3.6, page 28]
      PEXEnumTypeIndex  join_style; [§2.3.6, page 28]
} PEXLineBundleValues;
```

2.16.18. `PEXMarkerBundleEntry` Data Structure

```
typedef struct {
      PEXEnumTypeIndex  type; [§2.3.6, page 28]
      short             reserved;
      float             scale;
      PEXColorSpecifier color; [§2.5.14, page 32]
} PEXMarkerBundleEntry;
```

2.16.19. `PEXMarkerBundleValues` Data Structure

```
typedef struct {
      PEXEnumTypeIndex  type; [§2.3.6, page 28]
      short             reserved;
      float             scale;
      PEXColorSpecifier5_2 color; [§2.5.15, page 32]
} PEXMarkerBundleValues;
```

2.16.20. `PEXMarkerGlyphValues` Data Structure

```
typedef struct {
      PEXEnumTypeIndex  glyph_type; [§2.3.6, page 28]
      unsigned short    data_size; /* bytes of data following this field*/
      unsigned long     num_lines;
      PEXListOfCoord2D  *lines; [§2.7.6, page 33]
} PEXMarkerGlyphValues;
```

2.16.21. `PEXPatternEntry` Data Structure

```
typedef struct {
      PEXColorType      color_type; [§2.3.4, page 28]
      unsigned short    row_count;
      unsigned short    col_count;
      PEXArrayOfColor   colors; [§2.5.11, page 31]
} PEXPatternEntry;
```

2.16.22. `PEXPatternValues` Type Definition

```
typedef PEXPatternEntry PEXPatternValues;
```

2.16.23. `PEXTableInfo` Data Structure

```
typedef struct {
      unsigned short    definable_entries;
      unsigned short    predefined_count;
      unsigned short    predefined_min;
      unsigned short    predefined_max;
} PEXTableInfo;
```

2.16.24. `PEXTextBundleEntry` Data Structure

```
typedef struct {
      PEXTableIndex     font_index; [§2.3.21, page 29]
      PEXEnumTypeIndex  precision; [§2.3.6, page 28]
      float             char_expansion;
      float             char_spacing;
      PEXColorSpecifier color; [§2.5.14, page 32]
} PEXTextBundleEntry;
```

2.16.25. `PEXTextBundleValues` Data Structure

```
typedef struct {
      PEXTableIndex     font_index; [§2.3.21, page 29]
      PEXEnumTypeIndex  precision; [§2.3.6, page 28]
      float             char_expansion;
      float             char_spacing;
      PEXColorSpecifier5_2 color; [§2.5.15, page 32]
} PEXTextBundleValues;
```

2.16.26. `PEXTextFontEntry` Data Structure

```
typedef struct {
      unsigned short    count;
      PEXFont           *fonts; [§2.3.7, page 28]
} PEXTextFontEntry;
```

2.16.27. `PEXTextFontValues` Type Definition

```
typedef PEXTextFontEntry PEXTextFontValues;
```

2.16.28. `PEXTMValues` Data Structure

```
typedef struct {
      unsigned short          num_maps;
      unsigned short          reserved1;
      PEXTextureMap           *maps; [§2.3.22, page 29]
      PEXEnumTypeIndex        rendering_control; [§2.3.6, page 28]
      PEXEnumTypeIndex        parameterization_method; [§2.3.6, page 28]
      unsigned short          fp_data_index;
      unsigned short          reserved2;
      PEXMatrix               reflection_matrix; [§2.3.10, page 28]
      PEXMatrixHints          reflection_matrix_hints; [§2.3.11, page 28]
      PEXMatrix               orientation_matrix; [§2.3.10, page 28]
      PEXMatrixHints          orientation_matrix_hints; [§2.3.11,
page 28]
      PEXEnumTypeIndex        composition_method; [§2.3.6, page 28]
      unsigned short          channel_selector;
      PEXColorSpecifier5_2    environment_color; [§2.5.15, page 32]
      PEXColorSpecifier5_2    background_color; [§2.5.15, page 32]
      PEXEnumTypeIndex        min_sample_method; [§2.3.6, page 28]
      PEXEnumTypeIndex        max_sample_method; [§2.3.6, page 28]
```

```
        PEXEnumTypeIndex        t0_boundary_condition; [§2.3.6, page 28]
        PEXEnumTypeIndex        t1_boundary_condition; [§2.3.6, page 28]
        PEXEnumTypeIndex        t2_boundary_condition; [§2.3.6, page 28]
        PEXEnumTypeIndex        boundary_clamp_method; [§2.3.6, page 28]
        PEXColorSpecifier5_2    boundary_clamp_color; [§2.5.15, page 32]
        float                   depth_sampling_bias_hint;
        float                   t0_frequency_hint;
        float                   t1_frequency_hint;
        float                   t2_frequency_hint;
} PEXTMValues;
```

2.16.29. `PEXViewEntry` **Data Structure**

```
typedef struct {
        unsigned short    clip_flags;
        unsigned short    reserved1;
        PEXNPCSubVolume   clip_limits; [§2.26.7, page 62]
        PEXMatrix         orientation; [§2.3.10, page 28]
        PEXMatrix         mapping; [§2.3.10, page 28]
} PEXViewEntry;
```

2.16.30. `PEXViewValues` **Data Structure**

```
typedef struct {
        unsigned short    clip_flags;
        unsigned short    reserved1;
        PEXNPCSubVolume   clip_limits; [§2.26.7, page 62]
        PEXMatrix         orientation; [§2.3.10, page 28]
        PEXMatrixHints    orientation_hints; [§2.3.11, page 28]
        PEXMatrix         mapping; [§2.3.10, page 28]
        PEXMatrixHints    mapping_hints; [§2.3.11, page 28]
        PEXEnumTypeIndex  eyepoint_hint; [§2.3.6, page 28]
        unsigned short    reserved2;
        PEXCoord          eyepoint; [§2.7.1, page 33]
} PEXViewValues;
```

2.17. Data Structures for Match Rendering Targets

The data structures for match rendering targets include:

2.17.1. `PEXRenderingTarget` **Data Structure**

```
typedef struct {
        int         depth;
        int         type;
        Visual      *visual;
} PEXRenderingTarget;
```

2.18. Data Structures for Miscellaneous Data

The data structures for miscellaneous data include:

2.18.1. `PEXPointer` Data Structure

```
#if NeedFunctionPrototypes
typedef void *PEXPointer;
#else
typedef char *PEXPointer;
#endif
```

2.18.2. `PEXOCData` Data Structure

See PEXlib.h for this data structure, which is too long to reproduce here.

2.18.3. `PEXDMMColorRecord` Data Structure

```
typedef struct {
      unsigned int      count;      /* nbr of selectors that follow */
      unsigned short    *selectors;
} PEXDMMColorRecord;
```

2.19. Data Structures for Name Sets

The data structures for name sets include:

2.19.1. `PEXName` Data Structure

```
typedef unsigned long   PEXName;
```

2.19.2. `PEXNameSet` Data Structure

```
typedef XID             PEXNameSet;
```

2.19.3. `PEXNameSetPair` Data Structure

```
typedef struct {
      PEXNameSet        inclusion; [§2.19.2, page 46]
      PEXNameSet        exclusion; [§2.19.2, page 46]
} PEXNameSetPair;
```

2.19.4. `PEXListOfNameSetPair` Data Structure

```
typedef struct {
      unsigned short    count;      /* number of pairs */
      PEXNameSetPair    *pairs; [§2.19.3, page 46]
} PEXListOfNameSetPair;
```

2.20. Data Structures for NURBS

2.20.1. `PEXHomoColorCIE` Data Structure

```
typedef struct {
      float             x;
      float             y;
      float             z;
      float             w;
} PEXHomoColorCIE;
```

2.20.2. `PEXHomoColorHLS` Data Structure

```
typedef struct {
      float             hue;
      float             lightness;
      float             saturation;
      float             w;
} PEXHomoColorHLS;
```

2.20.3. `PEXHomoColorHSV` Data Structure

```
typedef struct {
      float             hue;
      float             saturation;
      float             value;
      float             w;
} PEXHomoColorHSV;
```

2.20.4. `PEXHomoColorRGB` Data Structure

```
typedef struct {
      float             red;
      float             green;
      float             blue;
      float             w;
} PEXHomoColorRGB;
```

2.20.5. `PEXHomoColorRGB8` Data Structure

```
typedef struct {
      unsigned char     red;
      unsigned char     green;
      unsigned char     blue;
      unsigned char     w;
} PEXHomoColorRGB8;
```

2.20.6. `PEXHomoColorRGB16` Data Structure

```
typedef struct {
      unsigned short    red;
      unsigned short    green;
      unsigned short    blue;
      unsigned short    w;
} PEXHomoColorRGB16;
```

2.20.7. `PEXCurveColorSpline` Data Structure

```
typedef struct {
      unsigned short    color_type;
      unsigned short    order;
      unsigned short    rationality;/* PEXRational or PEXNonRational */
      unsigned short    unused;
      unsigned int      num_knots;
      unsigned int      num_points;
      float             *knots;
      PEXPointer        colors;
} PEXCurveColorSpline;
```

2.20.8. PEXCurveGeometrySpline Data Structure

```
typedef struct {
      unsigned short    order;
      unsigned short    rationality;/* PEXRational or PEXNonRational */
      float             tmin;
      float             tmax;
      unsigned int      num_knots;
      unsigned int      num_points;
      float             *knots;
      PEXPointer        points; [§2.18.1, page 46]/* 3D or 4D PEXCoord */
} PEXCurveGeometrySpline;
```

2.20.9. PEXSurfaceColorSpline Data Structure

```
typedef struct {
      unsigned short    color_type;
      unsigned short    rationality;/* PEXRational or PEXNonRational */
      unsigned short    u_order;
      unsigned short    v_order;
      unsigned int      num_u_knots;
      unsigned int      num_v_knots;
      unsigned int      num_u_points;
      unsigned int      num_v_points;
      float             *u_knots;
      float             *v_knots;
      PEXPointer        colors;
} PEXSurfaceColorSpline;
```

2.20.10. PEXSurfaceDataSpline Data Structure

```
typedef struct {
      unsigned short    rationality;/* PEXRational or PEXNonRational */
      unsigned short    u_order;
      unsigned short    v_order;
      unsigned short    unused;
      unsigned int      num_u_knots;
      unsigned int      num_v_knots;
      unsigned int      num_u_points;
      unsigned int      num_v_points;
      float             *u_knots;
      float             *v_knots;
      unsigned int      num_floats;
      float             *points;
} PEXSurfaceDataSpline;
```

2.20.11. PEXSurfaceGeometrySpline Data Structure

```
typedef struct {
      unsigned short    rationality;/* PEXRational or PEXNonRational */
      unsigned short    uorder;
      unsigned short    vorder;
      unsigned short    unused;
      unsigned int      col_count;
      unsigned int      row_count;
      float             *uknots;
      float             *vknots;
      PEXPointer        points; [§2.18.1, page 46]/* 3D or 4D PEXCoord */
} PEXSurfaceGeometrySpline;
```

2.21. Data Structures for OC Context

The data structures for OC contexts include:

2.21.1. PEXOCC Data Structure

```
typedef struct _PEXOCC
#ifdef XLIB_ILLEGAL_ACCESS
{
      XExtData *ext_data;     /* hook for extension */
      /* PEXlib private data */
}
#endif
*PEXOCC;
```

2.21.2. PEXOCCValues Data Structure

```
typedef struct {
      Display           *display;
      PEXRenderer       renderer; [§2.3.17, page 29]
      PEXStructure      structure; [§2.3.19, page 29]
      PEXOCRequestType  req_type; [§2.3.14, page 28]
      int               shape_hint;
      int               ignore_edges;
      int               contour_hint;
      int               contours_all_one;
      unsigned int      facet_attributes;
      unsigned int      line_vertex_attributes;
      unsigned int      marker_vertex_attributes;
      unsigned int      surface_vertex_attributes;
      unsigned int      edge_attributes;
      unsigned int      facet_fp_data_count;
      unsigned int      line_vertex_fp_data_count;
```

```
        unsigned int      marker_vertex_fp_data_count;
        unsigned int      surface_vertex_fp_data_count;
        int               color_type;
        char              *encoding_state;
        int               data_model;
        union {
              PEXOCCStrideData        stride; [§2.21.3, page 50]
              PEXOCCUnpackedData      unpacked; [§2.21.4, page 50]
        } data_model_specs;
  } PEXOCCValues;
```

2.21.3. `PEXOCCStrideData` Data Structure

```
  typedef struct {
        int               facet_stride;
        int               vertex_stride;
        int               facet_color_offset;
        int               facet_normal_offset;
        int               facet_fp_data_offset;
        int               vertex_coord_offset;
        int               vertex_color_offset;
        int               vertex_normal_offset;
        int               vertex_edge_offset;
        int               vertex_radius_offset;
        int               vertex_axes_offset;
        int               vertex_angle_offset;
        int               line_vertex_fp_data_offset;
        int               marker_vertex_fp_data_offset;
        int               surface_vertex_fp_data_offset;
  } PEXOCCStrideData;
```

2.21.4. `PEXOCCUnpackedData` Data structure

```
  typedef struct {
        int               facet_color_size;
        int               facet_normal_size;
        int               facet_fp_data_count;
        int               vertex_coord_size;
        int               vertex_color_size;
        int               vertex_normal_size;
        int               vertex_edge_size;
        int               vertex_radius_size;
        int               vertex_axes_size;
        int               vertex_angle_size;
        int               line_vertex_fp_data_count;
        int               marker_vertex_fp_data_count;
        int               surface_vertex_fp_data_count;
  } PEXOCCUnpackedData;
```

2.21.5. `PEXListOfVertexData` Data Structure

```
  typedef struct {
        unsigned long     count;
        PEXPointer        vertices; [§2.18.1, page 46]
  } PEXListOfVertexData;
```

2.21.6. `PEXUnpackedFacetData` **Data Structure**

```
typedef struct {
      PEXPointer        colors; [§2.18.1, page 46]
      PEXPointer        normals; [§2.18.1, page 46]
      PEXPointer        fp_data; [§2.18.1, page 46]
} PEXUnpackedFacetData;
```

2.21.7. `PEXUnpackedVertexData` **Data Structure**

```
typedef struct {
      PEXPointer        coords; [§2.18.1, page 46]
      PEXPointer        colors; [§2.18.1, page 46]
      PEXPointer        normals; [§2.18.1, page 46]
      PEXPointer        edges; [§2.18.1, page 46]
      PEXPointer        radii; [§2.18.1, page 46]
      PEXPointer        axes; [§2.18.1, page 46]
      PEXPointer        angles; [§2.18.1, page 46]
      PEXPointer        fp_data; [§2.18.1, page 46]
} PEXUnpackedVertexData;
```

2.22. Data Structures for Parametric Surface Characteristics

The data structures for parametric surface characteristics include:

2.22.1. `PEXPSC` **Data Structure**

```
typedef struct {
      int               psc_type;
      PEXPSCData        characteristics; [§2.22.2, page 51]
} PEXPSC;
```

2.22.2. `PEXPSCData` **Data Structure**

```
typedef union {
      PEXPSCIsoparametricCurves     iso_curves; [§2.22.4, page 51]
      PEXPSCLevelCurves             level_curves; [§2.22.5, page 52]
      PEXPSCImpDepData              imp_dep; [§2.22.3, page 51]
} PEXPSCData;
```

2.22.3. `PEXPSCImpDepData` **Data Structure**

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

2.22.4. `PEXPSCIsoparametricCurves` **Data Structure**

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

2.22.5. PEXPSCLevelCurves Data Structure

```
typedef struct {
      PEXCoord          origin; [§2.7.1, page 33]
      PEXVector         direction; [§2.31.1, page 67]
      unsigned short    count;      /* number of parameters */
      unsigned short    reserved;
      float             *parameters;
} PEXPSCLevelCurves;
```

2.23. Data Structures for PHIGS Workstations

The data structures for PHIGS workstations include:

2.23.1. PEXListOfPostedStructure Data Structure

```
typedef struct {
      unsigned long           count;      /* number of posted structures*/
      PEXPostedStructure      *structures; [§2.23.5, page 52]
} PEXListOfPostedStructure;
```

2.23.2. PEXListOfView Data Structure

```
typedef struct {
      unsigned short    count;            /* number of views */
      PEXTableIndex     *views; [§2.3.21, page 29]
} PEXListOfView;
```

2.23.3. PEXPDAttributes Data Structure

```
typedef struct {
      unsigned short    status;
      PEXPickPath       path; [§2.25.7, page 60]
      int               path_order;
      PEXNameSet        inclusion; [§2.3.13, page 28]
      PEXNameSet        exclusion; [§2.3.13, page 28]
      PEXPickRecord     pick_record; [§2.25.8, page 60]
      PEXEnumTypeIndex  prompt_echo_type; [§2.3.6, page 28]
      PEXViewport       echo_volume; [§2.26.8, page 62]
      int               echo_switch;
} PEXPDAttributes;
```

2.23.4. PEXPMAttributes Data Structure

```
typedef struct {
    unsigned short      status;
    PEXPickPath         pick_path; [§2.25.7, page 60]
} PEXPMAttributes;
```

2.23.5. PEXPostedStructure Data Structure

```
typedef struct {
    PEXStructure        sid; [§2.3.19, page 29]
    float               priority;
} PEXPostedStructure;
```

2.23.6. PEXViewRep Data Structure

```
typedef struct {
    PEXTableIndex       index; [§2.3.21, page 29]
    unsigned short      reserved;
    PEXViewEntry        view; [§2.16.29, page 45]
} PEXViewRep;
```

2.23.7. PEXWorkstationAttributes Data Structure

```
typedef struct {
      short               drawable_update;
      int                 visual_state;
      int                 drawable_surface;
      int                 view_update;
      PEXListOfView       defined_views; [§2.23.2, page 52]
      int                 wks_update;
      PEXNPCSubVolume     req_npc_subvolume; [§2.26.7, page 62]
      PEXNPCSubVolume     cur_npc_subvolume; [§2.26.7, page 62]
      PEXViewport         req_workstation_viewport; [§2.26.8, page 62]
      PEXViewport         cur_workstation_viewport; [§2.26.8, page 62]
      int                 hlhsr_update;
      PEXEnumTypeIndex    req_hlhsr_mode; [§2.3.6, page 28]
      PEXEnumTypeIndex    cur_hlhsr_mode; [§2.3.6, page 28]
      Drawable            drawable;
      PEXLookupTable      marker_bundle; [§2.3.8, page 28]
      PEXLookupTable      text_bundle; [§2.3.8, page 28]
      PEXLookupTable      line_bundle; [§2.3.8, page 28]
      PEXLookupTable      interior_bundle; [§2.3.8, page 28]
      PEXLookupTable      edge_bundle; [§2.3.8, page 28]
      PEXLookupTable      color_table; [§2.3.8, page 28]
      PEXLookupTable      depth_cue_table; [§2.3.8, page 28]
      PEXLookupTable      light_table; [§2.3.8, page 28]
      PEXLookupTable      color_approx_table; [§2.3.8, page 28]
      PEXLookupTable      pattern_table; [§2.3.8, page 28]
      PEXLookupTable      text_font_table; [§2.3.8, page 28]
      PEXNameSet          highlight_incl; [§2.3.13, page 28]
      PEXNameSet          highlight_excl; [§2.3.13, page 28]
      PEXNameSet          invisibility_incl; [§2.3.13, page 28]
      PEXNameSet          invisibility_excl; [§2.3.13, page 28]
      PEXListOfPostedStructure      posted_structures; [§2.23.1, page 52]
      unsigned long       count_priorities;
      int                 buffer_update;
      int                 req_buffer_mode;
      int                 cur_buffer_mode;
} PEXWorkstationAttributes;
```

2.23.8. `PEXWorkstationDynamics` Data Structure

```
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;
```

2.24. Data Structures for Pipeline Context

The data structures for pipeline context include:

2.24.1. `PEXPCAttributes` Data Structure

```
typedef struct {
      PEXEnumTypeIndex        marker_type; [§2.3.6, page 28]
      float                   marker_scale;
      PEXColorSpecifier       marker_color; [§2.5.14, page 32]
      PEXTableIndex           marker_bundle_index; [§2.3.21, page 29]
      PEXTableIndex           text_font; [§2.3.21, page 29]
      unsigned short          text_precision;
      float                   char_expansion;
      float                   char_spacing;
      PEXColorSpecifier       text_color; [§2.5.14, page 32]
      float                   char_height;
      PEXVector2D             char_up_vector; [§2.31.2, page 67]
      unsigned short          text_path;
      PEXTextAlignment        text_alignment; [§2.24.13, page 59]
      float                   atext_height;
      PEXVector2D             atext_up_vector; [§2.31.2, page 67]
      unsigned short          atext_path;
      PEXTextAlignment        atext_alignment; [§2.24.13, page 59]
      PEXEnumTypeIndex        atext_style; [§2.3.6, page 28]
      PEXTableIndex           text_bundle_index; [§2.3.21, page 29]
      PEXEnumTypeIndex        line_type; [§2.3.6, page 28]
      float                   line_width;
      PEXColorSpecifier       line_color; [§2.5.14, page 32]
```

```
    PEXCurveApprox          curve_approx; [§2.24.3, page 58]
    PEXEnumTypeIndex        polyline_interp; [§2.3.6, page 28]
    PEXTableIndex           line_bundle_index; [§2.3.21, page 29]
    PEXEnumTypeIndex        interior_style; [§2.3.6, page 28]
    PEXTypeOrTableIndex     interior_style_index; [§2.3.23, page 29]
    PEXColorSpecifier       surface_color; [§2.5.14, page 32]
    PEXReflectionAttributes reflection_attr; [§2.24.10, page 58]
    PEXEnumTypeIndex        reflection_model; [§2.3.6, page 28]
    PEXEnumTypeIndex        surface_interp; [§2.3.6, page 28]
    PEXEnumTypeIndex        bf_interior_style; [§2.3.6, page 28]
    PEXTypeOrTableIndex     bf_interior_style_index; [§2.3.23, page 29]
    PEXColorSpecifier       bf_surface_color; [§2.5.14, page 32]
    PEXReflectionAttributes bf_reflection_attr; [§2.24.10, page 58]
    PEXEnumTypeIndex        bf_reflection_model; [§2.3.6, page 28]
    PEXEnumTypeIndex        bf_surface_interp; [§2.3.6, page 28]
    PEXSurfaceApprox        surface_approx; [§2.24.12, page 59]
    unsigned short          culling_mode;
    Bool                    distinguish;
    PEXCoord2D              pattern_size; [§2.7.2, page 33]
    PEXCoord                pattern_ref_point; [§2.7.1, page 33]
    PEXVector               pattern_ref_vec1; [§2.31.1, page 67]
    PEXVector               pattern_ref_vec2; [§2.31.1, page 67]
    PEXTableIndex           interior_bundle_index; [§2.3.21, page 29]
    PEXSwitch               surface_edges; [§2.3.20, page 29]
    PEXEnumTypeIndex        surface_edge_type; [§2.3.6, page 28]
    float                   surface_edge_width;
    PEXColorSpecifier       surface_edge_color; [§2.5.14, page 32]
    PEXTableIndex           edge_bundle_index; [§2.3.21, page 29]
    PEXMatrix               local_transform; [§2.3.10, page 28]
    PEXMatrix               global_transform; [§2.3.10, page 28]
    unsigned short          model_clip;
    PEXModelClipVolume      model_clip_volume; [§2.24.6, page 58]
    PEXTableIndex           view_index; [§2.3.21, page 29]
    PEXListOfLight          light_state; [§2.24.4, page 58]
    PEXTableIndex           depth_cue_index; [§2.3.21, page 29]
    PEXBitmask              asf_enables; [§2.3.3, page 28]
    PEXBitmask              asf_values; [§2.3.3, page 28]
    long                    pick_id;
    unsigned long           hlhsr_id;
    PEXNameSet              name_set; [§2.3.13, page 28]
    PEXTableIndex           color_approx_index; [§2.3.21, page 29]
    PEXEnumTypeIndex        rendering_color_model; [§2.3.6, page 28]
    PEXPSCSpecifier         para_surf_char; [§2.24.9, page 58]
} PEXPCAttributes;
```

2.24.2. `PEXPCAttributes5_2` Data Structure

```
typedef struct {
      PEXEnumTypeIndex        marker_type; [§2.3.6, page 28]
      float                   marker_scale;
      PEXColorSpecifier5_2    marker_color; [§2.5.15, page 32]
      PEXTableIndex           marker_bundle_index; [§2.3.21, page 29]
      PEXTableIndex           text_font; [§2.3.21, page 29]
      unsigned short          text_precision;
      float                   char_expansion;
      float                   char_spacing;
      PEXColorSpecifier5_2    text_color; [§2.5.15, page 32]
      float                   char_height;
      PEXVector2D             char_up_vector; [§2.31.2, page 67]
      unsigned short          text_path;
      PEXTextAlignment        text_alignment; [§2.24.13, page 59]
      float                   atext_height;
      PEXVector2D             atext_up_vector; [§2.31.2, page 67]
      unsigned short          atext_path;
      PEXTextAlignment        atext_alignment; [§2.24.13, page 59]
      PEXEnumTypeIndex        atext_style; [§2.3.6, page 28]
      PEXTableIndex           text_bundle_index; [§2.3.21, page 29]
      PEXEnumTypeIndex        line_type; [§2.3.6, page 28]
      float                   line_width;
      PEXColorSpecifier5_2    line_color; [§2.5.15, page 32]
      PEXCurveApprox          curve_approx; [§2.24.3, page 58]
      PEXEnumTypeIndex        polyline_interp; [§2.3.6, page 28]
      PEXTableIndex           line_bundle_index; [§2.3.21, page 29]
      PEXEnumTypeIndex        interior_style; [§2.3.6, page 28]
      PEXTypeOrTableIndex     interior_style_index; [§2.3.23, page 29]
      PEXColorSpecifier5_2    surface_color; [§2.5.15, page 32]
      PEXReflectionAttributes5_2
                        reflection_attr; [§2.24.11, page 59]
      PEXEnumTypeIndex        reflection_model; [§2.3.6, page 28]
      PEXEnumTypeIndex        surface_interp; [§2.3.6, page 28]
      PEXEnumTypeIndex        bf_interior_style; [§2.3.6, page 28]
      PEXTypeOrTableIndex     bf_interior_style_index; [§2.3.23, page 29]
      PEXColorSpecifier5_2    bf_surface_color; [§2.5.15, page 32]
      PEXReflectionAttributes5_2
                        bf_reflection_attr; [§2.24.11, page 59]
      PEXEnumTypeIndex        bf_reflection_model; [§2.3.6, page 28]
      PEXEnumTypeIndex        bf_surface_interp; [§2.3.6, page 28]
      PEXSurfaceApprox        surface_approx; [§2.24.12, page 59]
      unsigned short          culling_mode;
      Bool                    distinguish;
      PEXCoord2D              pattern_size; [§2.7.2, page 33]
      PEXCoord                pattern_ref_point; [§2.7.1, page 33]
      PEXVector               pattern_ref_vec1; [§2.31.1, page 67]
      PEXVector               pattern_ref_vec2; [§2.31.1, page 67]
      PEXTableIndex           interior_bundle_index; [§2.3.21, page 29]
      PEXSwitch               surface_edges; [§2.3.20, page 29]
      PEXEnumTypeIndex        surface_edge_type; [§2.3.6, page 28]
      float                   surface_edge_width;
      PEXColorSpecifier5_2    surface_edge_color; [§2.5.15, page 32]
      PEXTableIndex           edge_bundle_index; [§2.3.21, page 29]
```

```
    PEXMatrix                local_transform; [§2.3.10, page 28]
    PEXMatrix                global_transform; [§2.3.10, page 28]
    unsigned short           model_clip;
    PEXModelClipVolume       model_clip_volume; [§2.24.6, page 58]
    PEXTableIndex            view_index; [§2.3.21, page 29]
    PEXListOfLight           light_state; [§2.24.4, page 58]
    PEXTableIndex            depth_cue_index; [§2.3.21, page 29]
    PEXBitmask               asf_enables; [§2.3.3, page 28]
    PEXBitmask               asf_values; [§2.3.3, page 28]
    long                     pick_id;
    unsigned long            hlhsr_id;
    PEXNameSet               name_set; [§2.3.13, page 28]
    PEXTableIndex            color_approx_index; [§2.3.21, page 29]
    PEXEnumTypeIndex         rendering_color_model; [§2.3.6, page 28]
    PEXPSCSpecifier          para_surf_char; [§2.24.9, page 58]
    PEXEnumTypeIndex         alpha_blend_function; [§2.3.6, page 28]
    PEXTableIndex            bf_data_mapping_index; [§2.3.21, page 29]
    PEXDataMappingValues     bf_data_mapping_method; [§2.16.5, page 40]
    PEXTableIndex
                bf_interior_bundle_index; [§2.3.21, page 29]
    PEXTableIndex
                bf_reflectance_bundle_index; [§2.3.21, page 29]
    PEXBitmask               condition_flags; [§2.3.3, page 28]
    PEXTableIndex            data_mapping_index; [§2.3.21, page 29]
    PEXDataMappingValues     data_mapping_method; [§2.16.5, page 40]
    unsigned int             drawing_function;
    PEXBitmask               more_asf_enables[2];
    PEXBitmask               more_asf_values[2];
    PEXMatrixHints           global_transform_hints; [§2.3.11, page 28]
    PEXTableIndex            highlight_index; [§2.3.21, page 29]
    PEXEnumTypeIndex         line_cap_style; [§2.3.6, page 28]
    PEXEnumTypeIndex         line_join_style; [§2.3.6, page 28]
    float                    line_miter_limit;
    PEXMatrixHints           local_transform_hints; [§2.3.11, page 28]
    PEXTableIndex            para_surf_bundle_index; [§2.3.21, page 29]
    unsigned long            plane_mask;
    PEXPrimAntiAliasing      primitive_antialiasing; [§2.24.8, page 58]
    PEXTableIndex
                reflectance_bundle_index; [§2.3.21, page 29]
    Bool                     sequential_processing;
    int                      vertex_order;
    PEXEnumTypeIndex         z_buffer_test [§2.3.6, page 28]
    PEXListOfTexture         active_textures; [§2.24.4, page 58]
    PEXListOfTexture         bf_active_textures; [§2.24.4, page 58]
    PEXEnumTypeIndex
                tm_perspective_correction; [§2.3.6, page 28]
    PEXEnumTypeIndex         tm_resource_hints; [§2.3.6, page 28]
    PEXEnumTypeIndex         tm_sample_frequency; [§2.3.6, page 28]
} PEXPCAttributes5_2;
```

2.24.3. PEXCurveApprox Data Structure

```
typedef struct {
      PEXEnumTypeIndex  method; [§2.3.6, page 28]
      unsigned short    reserved;
      float             tolerance;
} PEXCurveApprox;
```

2.24.4. PEXListOfLight Data Structure

```
typedef struct {
      unsigned short    count;      /* number of lights */
      PEXTableIndex     *indices; [§2.3.21, page 29]
} PEXListOfLight;
```

2.24.5. PEXListOfTexture Data Structure

```
typedef struct {
      unsigned short    count;      /* number of lights */
      PEXTableIndex     *indices; [§2.3.21, page 29]
} PEXListOfTexture;
```

2.24.6. PEXModelClipVolume Data Structure

```
typedef struct {
      unsigned short    count;      /* number of half spaces */
      PEXHalfSpace      *half_spaces; [§2.14.1, page 39]
} PEXModelClipVolume;
```

2.24.7. PEXPipelineContext Data Structure

```
typedef XID       PEXPipelineContext;
```

2.24.8. PEXPrimAntiAliasing Data Structure

```
typedef struct {
      unsigned short    prim_types;
      PEXEnumTypeIndex  blend_function; [§2.3.6, page 28]
} PEXPrimAntiAliasing;
```

2.24.9. PEXPSCSpecifier Data Structure

```
typedef struct {
      short             type;
      PEXPSCData        psc; [§2.22.2, page 51]
} PEXPSCSpecifier;
```

2.24.10. PEXReflectionAttributes Data Structure

```
typedef struct {
      float             ambient;
      float             diffuse;
      float             specular;
      float             specular_conc;
      float             transmission;
      PEXColorSpecifier specular_color; [§2.5.14, page 32]
} PEXReflectionAttributes;
```

2.24.11. `PEXReflectionAttributes5_2` Data Structure

```
typedef struct {
      float                   ambient;
      float                   diffuse;
      float                   specular;
      float                   specular_conc;
      float                   transmission;
      PEXColorSpecifier5_2    specular_color; [§2.5.15, page 32]
} PEXReflectionAttributes5_2;
```

2.24.12. `PEXSurfaceApprox` Data Structure

```
typedef struct {
      PEXEnumTypeIndex  method; [§2.3.6, page 28]
      unsigned short    reserved;
      float             u_tolerance;
      float             v_tolerance;
} PEXSurfaceApprox;
```

2.24.13. `PEXTextAlignment` Data Structure

```
typedef struct {
      unsigned short    vertical;
      unsigned short    horizontal;
} PEXTextAlignment;
```

2.25. Data Structures for Rendering Picking

The data structures for rendering picking include:

2.25.1. `PEXMaxHitsReachedEvent` Data Structure

```
typedef struct {
      int               type;
      unsigned long     serial;
      Bool              send_event;
      Display           *display;
      PEXRenderer       renderer; [§2.3.17, page 29]
} PEXMaxHitsReachedEvent;
```

2.25.2. `PEXDeviceCoord2D` Data Structure

```
typedef struct {
      short             x;
      short             y;
} PEXDeviceCoord2D;
```

2.25.3. `PEXPDDCHitBox` Data Structure

```
typedef struct {
      PEXDeviceCoord2D  position; [§2.25.2, page 59]
      float             distance;
} PEXPDDCHitBox;
```

2.25.4. `PEXPDNPCHitVolume` Data Structure

```
typedef PEXNPCSubVolume       PEXPDNPCHitVolume;
```

2.25.5. `PEXPickDataRecord` **Data Structure**

```
typedef struct {
      unsigned short    length;     /* number of bytes in record */
      char              *record;
} PEXPickDataRecord;
```

2.25.6. `PEXPickElementRef` **Data Structure**

```
typedef struct {
      PEXStructure      sid; [§2.3.19, page 29]
      unsigned long     offset;
      unsigned long     pick_id;
} PEXPickElementRef;
```

2.25.7. `PEXPickPath` **Data Structure**

```
typedef struct {
      unsigned long     count;      /* number of elements */
      PEXPickElementRef *elements; [§2.25.6, page 60]
} PEXPickPath;
```

2.25.8. `PEXPickRecord` **Data Structure**

```
typedef union {
      PEXPDNPCHitVolume volume; [§2.25.4, page 59]
      PEXPDDCHitBox     box; [§2.25.3, page 59]
      PEXPickDataRecord data; [§2.25.5, page 60]
} PEXPickRecord;
```

2.26. Data Structures for Renderer Attributes

This section includes the renderer attributes data structure and the data structures it requires that are not defined elsewhere.

2.26.1. `PEXRendererAttributes` **Data Structure**

```
typedef struct {
      PEXPipelineContext      pipeline_context; [§2.3.16, page 28]
      PEXStructurePath        current_path; [§2.28.4, page 64]
      PEXLookupTable          marker_bundle; [§2.3.8, page 28]
      PEXLookupTable          text_bundle; [§2.3.8, page 28]
      PEXLookupTable          line_bundle; [§2.3.8, page 28]
      PEXLookupTable          interior_bundle; [§2.3.8, page 28]
      PEXLookupTable          edge_bundle; [§2.3.8, page 28]
      PEXLookupTable          view_table; [§2.3.8, page 28]
      PEXLookupTable          color_table; [§2.3.8, page 28]
      PEXLookupTable          depth_cue_table; [§2.3.8, page 28]
      PEXLookupTable          light_table; [§2.3.8, page 28]
      PEXLookupTable          color_approx_table; [§2.3.8, page 28]
      PEXLookupTable          pattern_table; [§2.3.8, page 28]
      PEXLookupTable          text_font_table; [§2.3.8, page 28]
      PEXNameSet              highlight_incl; [§2.19.2, page 46]
      PEXNameSet              highlight_excl; [§2.19.2, page 46]
      PEXNameSet              invisibility_incl; [§2.19.2, page 46]
      PEXNameSet              invisibility_excl; [§2.19.2, page 46]
      int                     renderer_state;
      PEXEnumTypeIndex        hlhsr_mode; [§2.3.6, page 28]
```

```
        PEXNPCSubVolume         npc_subvolume; [§2.26.7, page 62]
        PEXViewport             viewport; [§2.26.8, page 62]
        PEXListOfClipRect       clip_list; [§2.26.6, page 62]
        PEXNameSet              pick_incl; [§2.19.2, page 46]
        PEXNameSet              pick_excl; [§2.19.1, page 46]
        PEXStructurePath        pick_start_path; [§2.28.4, page 64]
        PEXColorSpecifier       background_color; [§2.5.14, page 32]
        Bool                    clear_image;
        Bool                    clear_z;
        int                     echo_mode;
} PEXRendererAttributes;
```

2.26.2. `PEXRendererAttributes5_2` Data Stucture

```
typedef struct {
        PEXPipelineContext      pipeline_context; [§2.3.16, page 28]
        PEXStructurePath        current_path; [§2.28.4, page 64]
        PEXLookupTable          marker_bundle; [§2.3.8, page 28]
        PEXLookupTable          text_bundle; [§2.3.8, page 28]
        PEXLookupTable          line_bundle; [§2.3.8, page 28]
        PEXLookupTable          interior_bundle; [§2.3.8, page 28]
        PEXLookupTable          edge_bundle; [§2.3.8, page 28]
        PEXLookupTable          view_table; [§2.3.8, page 28]
        PEXLookupTable          color_table; [§2.3.8, page 28]
        PEXLookupTable          depth_cue_table; [§2.3.8, page 28]
        PEXLookupTable          light_table; [§2.3.8, page 28]
        PEXLookupTable          color_approx_table; [§2.3.8, page 28]
        PEXLookupTable          pattern_table; [§2.3.8, page 28]
        PEXLookupTable          text_font_table; [§2.3.8, page 28]
        PEXNameSet              highlight_incl; [§2.19.2, page 46]
        PEXNameSet              highlight_excl; [§2.19.2, page 46]
        PEXNameSet              invisibility_incl; [§2.19.2, page 46]
        PEXNameSet              invisibility_excl; [§2.19.2, page 46]
        int                     renderer_state;
        PEXEnumTypeIndex        hlhsr_mode; [§2.3.6, page 28]
        PEXNPCSubVolume         npc_subvolume; [§2.26.7, page 62]
        PEXViewport             viewport; [§2.26.8, page 62]
        PEXListOfClipRect       clip_list; [§2.26.6, page 62]
        PEXNameSet              pick_incl; [§2.19.2, page 46]
        PEXNameSet              pick_excl; [§2.19.1, page 46]
        PEXStructurePath        pick_start_path; [§2.28.4, page 64]
        PEXColorSpecifier5_2    background_color; [§2.5.15, page 32]
        Bool                    clear_image;
        Bool                    clear_z;
        int                     echo_mode;
        float                   clear_z_value;
        Bool                    clear_alpha;
        Bool                    phigs_interior_bundle;
        PEXLookupTable          extent_size_table; [§2.3.8, page 28]
```

```
        PEXLookupTable          data_mapping_table; [§2.3.8, page 28]
        PEXLookupTable          highlight_table; [§2.3.8, page 28]
        PEXLookupTable          marker_glyph_table; [§2.3.8, page 28]
        PEXEnumTypeIndex        transparency_method; [§2.3.6, page 28]
        PEXListOfFloat          transparency_thresholds; [§2.30.1, page 67]
        PEXSwitch               pixel_color_mode; [§2.3.20, page 29]
        PEXColorSpecifier5_2    echo_color; [§2.5.15, page 32]
        PEXEnumTypeIndex        echo_method; [§2.3.6, page 28]
        PEXLookupTable          texture_mapping_table; [§2.3.8, page 28]
} PEXRendererAttributes5_2;
```

2.26.3. `PEXDeviceCoord` Data Structure

```
typedef struct {
        short                   x;
        short                   y;
        float                   z;
} PEXDeviceCoord;
```

2.26.4. `PEXDeviceRect` Data Structure

```
typedef struct {
        short                   xmin;
        short                   ymin;
        short                   xmax;
        short                   ymax;
} PEXDeviceRect;
```

2.26.5. `PEXElementRef` Data Structure

```
typedef struct {
        PEXStructure            structure; [§2.3.19, page 29]
        unsigned long           offset;
} PEXElementRef;
```

2.26.6. `PEXListOfClipRect` Data Structure

```
typedef struct {
        unsigned short          count;
        PEXDeviceRect           *rectangles; [§2.26.4, page 62]
} PEXListOfClipRect;
```

2.26.7. `PEXNPCSubVolume` Data Structure

```
typedef struct {
        PEXCoord                min; [§2.7.1, page 33]
        PEXCoord                max; [§2.7.1, page 33]
} PEXNPCSubVolume;
```

2.26.8. `PEXViewport` Data Structure

```
typedef struct {
        PEXDeviceCoord          min; [§2.26.3, page 62]
        PEXDeviceCoord          max; [§2.26.3, page 62]
        PEXSwitch               use_drawable; [§2.3.20, page 29]
        unsigned char           reserved[3];
} PEXViewport;
```

2.27. Data Structures for Search Context

The data structures for search context include:

2.27.1. `PEXSCAttributes` Data Structure

```
typedef struct {
      PEXCoord                position; [§2.7.1, page 33]
      float                   distance;
      unsigned short          ceiling;
      Bool                    model_clip_flag;
      PEXStructurePath        start_path; [§2.28.4, page 64]
      PEXListOfNameSetPair    normal; [§2.19.4, page 46]
      PEXListOfNameSetPair    inverted; [§2.19.4, page 46]
} PEXSCAttributes;
```

2.27.2. `PEXSCAttributes5_2` Data Structure

```
typedef struct {
      PEXCoord                position; [§2.7.1, page 33]
      float                   distance;
      unsigned short          ceiling;
      Bool                    model_clip_flag;
      PEXStructurePath        start_path; [§2.28.4, page 64]
      PEXListOfNameSetPair    normal; [§2.19.4, page 46]
      PEXListOfNameSetPair    inverted; [§2.19.4, page 46]
      PEXLookupTable          text_font_table;
} PEXSCAttributes5_2;
```

2.28. Data Structures for Structures

The data structures for structures include:

2.28.1. `PEXElementInfo` Data Structure

```
typedef struct {
      unsigned short    type;
      unsigned short    length;
} PEXElementInfo;
```

2.28.2. `PEXLargeElementInfo` Data Structure

```
typedef struct {
      unsigned int      type;
      unsigned int      length;
} PEXLargeElementInfo;
```

2.28.3. `PEXStructureInfo` Data Structure

```
typedef struct {
      unsigned long     element_count;
      unsigned long     size;
      Bool              has_refs;
      unsigned short    edit_mode;
      unsigned long     element_pointer;
      unsigned short    structure_permission;
} PEXStructureInfo;
```

2.28.4. `PEXStructurePath` Data Structure

```
typedef struct {
      unsigned long     count;
      PEXElementRef     *elements; [§2.26.5, page 62]
} PEXStructurePath;
```

2.29. Data Structures for Textures

The data structures for textures include:

2.29.1. `PEXTexelLum` Data Structure

```
typedef struct  {
      float             luminance;      /* Valid range : [0.0,1.0]      */
} PEXTexelLum;
```

2.29.2. `PEXTexelLum8` Data Structure

```
typedef struct {
      unsigned char     luminance;      /* Valid range : [0,255]        */
} PEXTexelLum8;
```

2.29.3. `PEXTexelLum16` Data Structure

```
typedef struct{
      unsigned short    luminance;      /* Valid range : [0,65535]      */
} PEXTexelLum16;
```

2.29.4. `PEXTexelLumA` Data Structure

```
typedef struct {
      float             luminance;      /* Valid range : [0.0,1.0]      */
      float             alpha;          /* Valid range : [0.0,1.0]      */
} PEXTexelLumA;
```

2.29.5. `PEXTexelLumA8` Data Structure

```
typedef struct {
      unsigned char     luminance;      /* Valid range : [0,255]        */
      unsigned char     alpha;          /* Valid range : [0,255]        */
} PEXTexelLumA8;
```

2.29.6. `PEXTexelLumA16` Data Structure

```
typedef struct {
      unsigned short    luminance;      /* Valid range : [0,65535]      */
      unsigned short    alpha;          /* Valid range : [0,65535]      */
} PEXTexelLumA16;
```

2.29.7. `PEXTexelRGB` Data Structure

```
typedef struct {
      float             red;
      float             green;          /* Valid range : [0.0,1.0]      */
      float             blue;
} PEXTexelRGB;
```

2.29.8. `PEXTexelRGB8` Data Structure

```
typedef struct {
      unsigned char     red;
      unsigned char     green;          /* Valid range : [0,255]        */
      unsigned char     blue;
} PEXTexelRGB8;
```

2.29.9. `PEXTexelRGB16` Data Structure

```
typedef struct {
      unsigned short    red;
      unsigned short    green;          /* Valid range : [0,65535]      */
      unsigned short    blue;
} PEXTexelRGB16;
```

2.29.10. `PEXTexelRGBA` Data Structure

```
typedef struct {
      float             red;
      float             green;          /* Valid range : [0.0,1.0]      */
      float             blue;
      float             alpha;
} PEXTexelRGBA;
```

2.29.11. `PEXTexelRGBA8` Data Structure

```
typedef struct {
      unsigned char     red;
      unsigned char     green;          /* Valid range : [0,255]        */
      unsigned char     blue;
      unsigned char     alpha;
} PEXTexelRGBA8;
```

2.29.12. `PEXTexelRGBA16` Data Structure

```
typedef struct {
      unsigned short    red;
      unsigned short    green;          /* Valid range : [0,65535]      */
      unsigned short    blue;
      unsigned short    alpha;
} PEXTexelRGBA16;
```

2.29.13. `PEXTexelDimension` Data Structure

```
typedef struct {
      unsigned short int      c0;
      unsigned short int      c1;
      unsigned short int      c2;
} PEXTexelDimension;
```

2.29.14. PEXMipmapLevel Data Structure

```
typedef struct {
      PEXTexelDimension       dimension; [§2.29.13, page 65]
      union {
            PEXTexelLum             *luminance; [§2.29.1, page 64]
            PEXTexelLum8            *luminance8; [§2.29.2, page 64]
            PEXTexelLum16           *luminance16; [§2.29.3, page 64]
            PEXTexelLumA            *luminance_alpha; [§2.29.4, page 64]
            PEXTexelLumA8           *luminance_alpha8; [§2.29.5, page 64]
            PEXTexelLumA16          *luminance_alpha16; [§2.29.6,
page 64]
            PEXTexelRGB             *rgb; [§2.29.7, page 64]
            PEXTexelRGB8            *rgb8; [§2.29.8, page 65]
            PEXTexelRGB16           *rgb16; [§2.29.9, page 65]
            PEXTexelRGBA            *rgb_alpha; [§2.29.10, page 65]
            PEXTexelRGBA8           *rgb_alpha8; [§2.29.11, page 65]
            PEXTexelRGBA16          *rgb_alpha16; [§2.29.12, page 65]
      } array;
} PEXMipmapLevel;
```

2.29.15. PEXTMCoordData Data Structure

```
typedef struct {
      PEXEnumTypeIndex              projection; [§2.3.6, page 28]
      PEXTMParameterizationData     param_data; [§2.29.17, page 66]
      PEXEnumTypeIndex              coord_source; [§2.3.6, page 28]
      unsigned short int            fp_data_index;
      PEXMatrix                     mc_transform; [§2.3.10, page 28]
} PEXTMCoordData;
```

2.29.16. PEXTMDomainData Data Structure

```
typedef struct {
      PEXEnumTypeIndex        tm_type; [§2.3.6, page 28]
      PEXEnumTypeIndex        texel_type; [§2.3.6, page 28]
      unsigned int            num_levels;
} PEXTMDomainData;
```

2.29.17. PEXTMParameterizationData Data Structure

```
typedef struct {
      union {
            struct {
                  PEXMatrix   matrix; [§2.3.10, page 28]
            } reflection;
            struct {
                  float       p0[4];
                  float       p1[4];
            } linear;
      } data;
} PEXTMParameterizationData;
```

2.30. Data Structures for Trimming Curves

The data structures for trimming curves include:

2.30.1. `PEXListOfFloat` Data Structure

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

2.30.2. `PEXListOfTrimCurve` Data Structure

```
typedef struct {
      unsigned short    count;                  /* number of curves */
      PEXTrimCurve      *curves; [§2.30.3, page 67]
} PEXListOfTrimCurve;
```

2.30.3. `PEXTrimCurve` Data Structure

```
typedef struct {
      PEXSwitch         visibility; [§2.3.20, page 29]
      unsigned char     reserved;
      unsigned short    order;
      PEXCoordType      rationality; [§2.3.5, page 28]
      PEXEnumTypeIndex  approx_method; [§2.3.6, page 28]
      float             tolerance;
      float             tmin, tmax;
      PEXListOfFloat    knots; [§2.30.1, page 67]
      unsigned short    count;
      PEXArrayOfCoord   control_points; [§2.7.4, page 33]
} PEXTrimCurve;
```

2.31. Data Structures for Vectors

The data structures for vectors include:

2.31.1. `PEXVector` Data Structure

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

2.31.2. `PEXVector2D` Data Structure

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

2.32. Data Structures for Vertex Data

The data structures for vertex data include:

2.32.1. `PEXVertexIndexed` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorIndexed         index; [§2.5.1, page 30]
} PEXVertexIndexed;
```

2.32.2. `PEXVertexCIE` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorCIE             cie; [§2.5.2, page 30]
} PEXVertexCIE;
```

2.32.3. `PEXVertexHLS` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorHLS             hls; [§2.5.3, page 30]
} PEXVertexHLS;
```

2.32.4. `PEXVertexHSV` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorHSV             hsv; [§2.5.4, page 30]
} PEXVertexHSV;
```

2.32.5. `PEXVertexRGB` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGB             rgb; [§2.5.5, page 30]
} PEXVertexRGB;
```

2.32.6. `PEXVertexRGBA` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGBA            rgba; [§2.5.6, page 31]
} PEXVertexRGBA;
```

2.32.7. `PEXVertexRGB8` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGB8            rgb8; [§2.5.7, page 31]
} PEXVertexRGB8;
```

2.32.8. `PEXVertexRGBA8` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGBA8           rgba8; [§2.5.8, page 31]
} PEXVertexRGBA8;
```

2.32.9. `PEXVertexRGB16` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGB16           rgb16; [§2.5.9, page 31]
} PEXVertexRGB16;
```

2.32.10. `PEXVertexRGBA16` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGBA16          rgba16; [§2.5.10, page 31]
} PEXVertexRGBA16;
```

2.32.11. `PEXVertexNormal` Data Structure

```
typedef struct {
      PEXCoord              point; [§2.7.1, page 33]
      PEXVector             normal; [§2.31.1, page 67]
} PEXVertexNormal;
```

2.32.12. `PEXVertexIndexedNormal` Data Structure

```
typedef struct {
      PEXCoord              point; [§2.7.1, page 33]
      PEXColorIndexed       index; [§2.5.1, page 30]
      PEXVector             normal; [§2.31.1, page 67]
} PEXVertexIndexedNormal;
```

2.32.13. `PEXVertexCIENormal` Data Structure

```
typedef struct {
      PEXCoord              point; [§2.7.1, page 33]
      PEXColorCIE           cie; [§2.5.2, page 30]
      PEXVector             normal; [§2.31.1, page 67]
} PEXVertexCIENormal;
```

2.32.14. `PEXVertexHLSNormal` Data Structure

```
typedef struct {
      PEXCoord              point; [§2.7.1, page 33]
      PEXColorHLS           hls; [§2.5.3, page 30]
      PEXVector             normal; [§2.31.1, page 67]
} PEXVertexHLSNormal;
```

2.32.15. `PEXVertexHSVNormal` Data Structure

```
typedef struct {
      PEXCoord              point; [§2.7.1, page 33]
      PEXColorHSV           hsv; [§2.5.4, page 30]
      PEXVector             normal; [§2.31.1, page 67]
} PEXVertexHSVNormal;
```

2.32.16. `PEXVertexRGBNormal` Data Structure

```
typedef struct {
      PEXCoord              point; [§2.7.1, page 33]
      PEXColorRGB           rgb; [§2.5.5, page 30]
      PEXVector             normal; [§2.31.1, page 67]
} PEXVertexRGBNormal;
```

2.32.17. `PEXVertexRGBANormal` Data Structure

```
typedef struct {
      PEXCoord              point; [§2.7.1, page 33]
      PEXColorRGBA          rgba; [§2.5.6, page 31]
      PEXVector             normal; [§2.31.1, page 67]
} PEXVertexRGBANormal;
```

2.32.18. PEXVertexRGB8Normal Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGB8            rgb8; [§2.5.7, page 31]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexRGB8Normal;
```

2.32.19. PEXVertexRGBA8Normal Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGBA8           rgba8; [§2.5.8, page 31]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexRGBA8Normal;
```

2.32.20. PEXVertexRGB16Normal Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGB16           rgb16; [§2.5.9, page 31]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexRGB16Normal;
```

2.32.21. PEXVertexRGBA16Normal Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGBA16          rgba16; [§2.5.10, page 31]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexRGBA16Normal;
```

2.32.22. PEXVertexEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      unsigned int            edge;
} PEXVertexEdge;
```

2.32.23. PEXVertexIndexedEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorIndexed         index; [§2.5.1, page 30]
      unsigned int            edge;
} PEXVertexIndexedEdge;
```

2.32.24. PEXVertexCIEEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorCIE             cie; [§2.5.2, page 30]
      unsigned int            edge;
} PEXVertexCIEEdge;
```

2.32.25. PEXVertexHLSEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorHLS             hls; [§2.5.3, page 30]
      unsigned int            edge;
} PEXVertexHLSEdge;
```

2.32.26. PEXVertexHSVEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorHSV             hsv; [§2.5.4, page 30]
      unsigned int            edge;
} PEXVertexHSVEdge;
```

2.32.27. PEXVertexRGBEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGB             rgb; [§2.5.5, page 30]
      unsigned int            edge;
} PEXVertexRGBEdge;
```

2.32.28. PEXVertexRGBAEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGBA            rgba; [§2.5.6, page 31]
      unsigned int            edge;
} PEXVertexRGBAEdge;
```

2.32.29. PEXVertexRGB8Edge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGB8            rgb8; [§2.5.7, page 31]
      unsigned int            edge;
} PEXVertexRGB8Edge;
```

2.32.30. PEXVertexRGBA8Edge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGBA8           rgba8; [§2.5.8, page 31]
      unsigned int            edge;
} PEXVertexRGBA8Edge;
```

2.32.31. PEXVertexRGB16Edge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGB16           rgb16; [§2.5.9, page 31]
      unsigned int            edge;
} PEXVertexRGB16Edge;
```

2.32.32. PEXVertexRGBA16Edge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGBA16          rgba16; [§2.5.10, page 31]
      unsigned int            edge;
} PEXVertexRGBA16Edge;
```

2.32.33. PEXVertexNormalEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXVector               normal; [§2.31.1, page 67]
      unsigned int            edge;
} PEXVertexNormalEdge;
```

2.32.34. PEXVertexIndexedNormalEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorIndexed         index; [§2.5.1, page 30]
      PEXVector               normal; [§2.31.1, page 67]
      unsigned int            edge;
} PEXVertexIndexedNormalEdge;
```

2.32.35. PEXVertexCIENormalEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorCIE             cie; [§2.5.2, page 30]
      PEXVector               normal; [§2.31.1, page 67]
      unsigned int            edge;
} PEXVertexCIENormalEdge;
```

2.32.36. PEXVertexHLSNormalEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorHLS             hls; [§2.5.3, page 30]
      PEXVector               normal; [§2.31.1, page 67]
      unsigned int            edge;
} PEXVertexHLSNormalEdge;
```

2.32.37. PEXVertexHSVNormalEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorHSV             hsv; [§2.5.4, page 30]
      PEXVector               normal; [§2.31.1, page 67]
      unsigned int            edge;
} PEXVertexHSVNormalEdge;
```

2.32.38. PEXVertexRGBNormalEdge Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGB             rgb; [§2.5.5, page 30]
      PEXVector               normal; [§2.31.1, page 67]
      unsigned int            edge;
} PEXVertexRGBNormalEdge;
```

2.32.39. `PEXVertexRGBANormalEdge` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGBA            rgba; [§2.5.6, page 31]
      PEXVector               normal; [§2.31.1, page 67]
      unsigned int            edge;
} PEXVertexRGBANormalEdge;
```

2.32.40. `PEXVertexRGB8NormalEdge` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGB8            rgb8; [§2.5.7, page 31]
      PEXVector               normal; [§2.31.1, page 67]
      unsigned int            edge;
} PEXVertexRGB8NormalEdge;
```

2.32.41. `PEXVertexRGBA8NormalEdge` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGBA8           rgba8; [§2.5.8, page 31]
      PEXVector               normal; [§2.31.1, page 67]
      unsigned int            edge;
} PEXVertexRGBA8NormalEdge;
```

2.32.42. `PEXVertexRGB16NormalEdge` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGB16           rgb16; [§2.5.9, page 31]
      PEXVector               normal; [§2.31.1, page 67]
      unsigned int            edge;
} PEXVertexRGB16NormalEdge;
```

2.32.43. `PEXVertexRGBA16NormalEdge` Data Structure

```
typedef struct {
      PEXCoord                point; [§2.7.1, page 33]
      PEXColorRGBA16          rgba16; [§2.5.10, page 31]
      PEXVector               normal; [§2.31.1, page 67]
      unsigned int            edge;
} PEXVertexRGBA16NormalEdge;
```

2.32.44. `PEXVertexIndexed2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorIndexed         index; [§2.5.1, page 30]
} PEXVertexIndexed2D;
```

2.32.45. `PEXVertexCIE2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorCIE             cie; [§2.5.2, page 30]
} PEXVertexCIE2D;
```

2.32.46. `PEXVertexHLS2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorHLS             hls; [§2.5.3, page 30]
} PEXVertexHLS2D;
```

2.32.47. `PEXVertexHSV2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorHSV             hsv; [§2.5.4, page 30]
} PEXVertexHSV2D;
```

2.32.48. `PEXVertexRGB2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGB             rgb; [§2.5.5, page 30]
} PEXVertexRGB2D;
```

2.32.49. `PEXVertexRGBA2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGBA            rgba; [§2.5.6, page 31]
} PEXVertexRGBA2D;
```

2.32.50. `PEXVertexRGB82D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGB8            rgb8; [§2.5.7, page 31]
} PEXVertexRGB82D;
```

2.32.51. `PEXVertexRGBA82D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGBA8           rgba8; [§2.5.8, page 31]
} PEXVertexRGBA82D;
```

2.32.52. `PEXVertexRGB162D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGB16           rgb16; [§2.5.9, page 31]
} PEXVertexRGB162D;
```

2.32.53. `PEXVertexRGBA162D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGBA16          rgba16; [§2.5.10, page 31]
} PEXVertexRGBA162D;
```

2.32.54. `PEXVertexNormal2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexNormal2D;
```

2.32.55. `PEXVertexIndexedNormal2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorIndexed         index; [§2.5.1, page 30]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexIndexedNormal2D;
```

2.32.56. `PEXVertexCIENormal2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorCIE             cie; [§2.5.2, page 30]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexCIENormal2D;
```

2.32.57. `PEXVertexHLSNormal2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorHLS             hls; [§2.5.3, page 30]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexHLSNormal2D;
```

2.32.58. `PEXVertexHSVNormal2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorHSV             hsv; [§2.5.4, page 30]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexHSVNormal2D;
```

2.32.59. `PEXVertexRGBNormal2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGB             rgb; [§2.5.5, page 30]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexRGBNormal2D;
```

2.32.60. `PEXVertexRGBANormal2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGBA            rgba; [§2.5.6, page 31]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexRGBANormal2D;
```

2.32.61. `PEXVertexRGB8Normal2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGB8            rgb8; [§2.5.7, page 31]
      PEXVector               normal; [§2.31.1, page 67]
} PEXVertexRGB8Normal2D;
```

2.32.62. `PEXVertexRGBA8Normal2D` Data Structure

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      PEXColorRGBA8         rgba8; [§2.5.8, page 31]
      PEXVector             normal; [§2.31.1, page 67]
} PEXVertexRGBA8Normal2D;
```

2.32.63. `PEXVertexRGB16Normal2D` Data Structure

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      PEXColorRGB16         rgb16; [§2.5.9, page 31]
      PEXVector             normal; [§2.31.1, page 67]
} PEXVertexRGB16Normal2D;
```

2.32.64. `PEXVertexRGBA16Normal2D` Data Structure

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      PEXColorRGBA16        rgba16; [§2.5.10, page 31]
      PEXVector             normal; [§2.31.1, page 67]
} PEXVertexRGBA16Normal2D;
```

2.32.65. `PEXVertexEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      unsigned int          edge;
} PEXVertexEdge2D;
```

2.32.66. `PEXVertexIndexedEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      PEXColorIndexed       index; [§2.5.1, page 30]
      unsigned int          edge;
} PEXVertexIndexedEdge2D;
```

2.32.67. `PEXVertexCIEEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      PEXColorCIE           cie; [§2.5.2, page 30]
      unsigned int          edge;
} PEXVertexCIEEdge2D;
```

2.32.68. `PEXVertexHLSEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      PEXColorHLS           hls; [§2.5.3, page 30]
      unsigned int          edge;
} PEXVertexHLSEdge2D;
```

2.32.69. `PEXVertexHSVEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorHSV             hsv; [§2.5.4, page 30]
      unsigned int            edge;
} PEXVertexHSVEdge2D;
```

2.32.70. `PEXVertexRGBEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGB             rgb; [§2.5.5, page 30]
      unsigned int            edge;
} PEXVertexRGBEdge2D;
```

2.32.71. `PEXVertexRGBAEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGBA            rgba; [§2.5.6, page 31]
      unsigned int            edge;
} PEXVertexRGBAEdge2D;
```

2.32.72. `PEXVertexRGB8Edge2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGB8            rgb8; [§2.5.7, page 31]
      unsigned int            edge;
} PEXVertexRGB8Edge2D;
```

2.32.73. `PEXVertexRGBA8Edge2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGBA8           rgba8; [§2.5.8, page 31]
      unsigned int            edge;
} PEXVertexRGBA8Edge2D;
```

2.32.74. `PEXVertexRGB16Edge2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGB16           rgb16; [§2.5.9, page 31]
      unsigned int            edge;
} PEXVertexRGB16Edge2D;
```

2.32.75. `PEXVertexRGBA16Edge2D` Data Structure

```
typedef struct {
      PEXCoord2D              point; [§2.7.2, page 33]
      PEXColorRGBA16          rgba16; [§2.5.10, page 31]
      unsigned int            edge;
} PEXVertexRGBA16Edge2D;
```

2.32.76. `PEXVertexNormalEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D               point; [§2.7.2, page 33]
      PEXVector                normal; [§2.31.1, page 67]
      unsigned int             edge;
} PEXVertexNormalEdge2D;
```

2.32.77. `PEXVertexIndexedNormalEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D               point; [§2.7.2, page 33]
      PEXColorIndexed          index; [§2.5.1, page 30]
      PEXVector                normal; [§2.31.1, page 67]
      unsigned int             edge;
} PEXVertexIndexedNormalEdge2D;
```

2.32.78. `PEXVertexCIENormalEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D               point; [§2.7.2, page 33]
      PEXColorCIE              cie; [§2.5.2, page 30]
      PEXVector                normal; [§2.31.1, page 67]
      unsigned int             edge;
} PEXVertexCIENormalEdge2D;
```

2.32.79. `PEXVertexHLSNormalEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D               point; [§2.7.2, page 33]
      PEXColorHLS              hls; [§2.5.3, page 30]
      PEXVector                normal; [§2.31.1, page 67]
      unsigned int             edge;
} PEXVertexHLSNormalEdge2D;
```

2.32.80. `PEXVertexHSVNormalEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D               point; [§2.7.2, page 33]
      PEXColorHSV              hsv; [§2.5.4, page 30]
      PEXVector                normal; [§2.31.1, page 67]
      unsigned int             edge;
} PEXVertexHSVNormalEdge2D;
```

2.32.81. `PEXVertexRGBNormalEdge2D` Data Structure

```
typedef struct {
      PEXCoord2D               point; [§2.7.2, page 33]
      PEXColorRGB              rgb; [§2.5.5, page 30]
      PEXVector                normal; [§2.31.1, page 67]
      unsigned int             edge;
} PEXVertexRGBNormalEdge2D;
```

2.32.82. `PEXVertexRGBANormalEdge2D` **Data Structure**

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      PEXColorRGBA          rgba; [§2.5.6, page 31]
      PEXVector             normal; [§2.31.1, page 67]
      unsigned int          edge;
} PEXVertexRGBANormalEdge2D;
```

2.32.83. `PEXVertexRGB8NormalEdge2D` **Data Structure**

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      PEXColorRGB8          rgb8; [§2.5.7, page 31]
      PEXVector             normal; [§2.31.1, page 67]
      unsigned int          edge;
} PEXVertexRGB8NormalEdge2D;
```

2.32.84. `PEXVertexRGBA8NormalEdge2D` **Data Structure**

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      PEXColorRGBA8         rgba8; [§2.5.8, page 31]
      PEXVector             normal; [§2.31.1, page 67]
      unsigned int          edge;
} PEXVertexRGBA8NormalEdge2D;
```

2.32.85. `PEXVertexRGB16NormalEdge2D` **Data Structure**

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      PEXColorRGB16         rgb16; [§2.5.9, page 31]
      PEXVector             normal; [§2.31.1, page 67]
      unsigned int          edge;
} PEXVertexRGB16NormalEdge2D;
```

2.32.86. `PEXVertexRGBA16NormalEdge2D` **Data Structure**

```
typedef struct {
      PEXCoord2D            point; [§2.7.2, page 33]
      PEXColorRGBA16        rgba16; [§2.5.10, page 31]
      PEXVector             normal; [§2.31.1, page 67]
      unsigned int          edge;
} PEXVertexRGBA16NormalEdge2D;
```

2.32.87. `PEXArrayOfVertex` **Data Structure**

```
typedef union {
      PEXCoord                    *no_data; [§2.7.1, page 33]
      PEXVertexIndexed            *index; [§2.5.1, page 30]
      PEXVertexCIE                *cie; [§2.5.2, page 30]
      PEXVertexHLS                *hls; [§2.5.3, page 30]
      PEXVertexHSV                *hsv; [§2.5.4, page 30]
      PEXVertexRGB                *rgb; [§2.5.5, page 30]
      PEXVertexRGBA               *rgba; [§2.5.6, page 31]
      PEXVertexRGB8               *rgb8; [§2.5.7, page 31]
      PEXVertexRGBA8              *rgba8; [§2.5.8, page 31]
      PEXVertexRGB16              *rgb16; [§2.5.9, page 31]
      PEXVertexRGBA16             *rgba16; [§2.5.10, page 31]
```

```
      PEXVertexNormal             *normal; [§2.32.11, page 69]
      PEXVertexEdge               *edge; [§2.32.22, page 70]
      PEXVertexIndexedNormal      *index_normal; [§2.32.12, page 69]
      PEXVertexCIENormal          *cie_normal; [§2.32.13, page 69]
      PEXVertexHLSNormal          *hls_normal; [§2.32.14, page 69]
      PEXVertexHSVNormal          *hsv_normal; [§2.32.15, page 69]
      PEXVertexRGBNormal          *rgb_normal; [§2.32.16, page 69]
      PEXVertexRGBANormal         *rgba_normal; [§2.32.17, page 69]
      PEXVertexRGB8Normal         *rgb8_normal; [§2.32.18, page 70]
      PEXVertexRGBA8Normal        *rgba8_normal; [§2.32.19, page 70]
      PEXVertexRGB16Normal        *rgb16_normal; [§2.32.20, page 70]
      PEXVertexRGBA16Normal       *rgba16_normal; [§2.32.21, page 70]
      PEXVertexIndexedEdge        *index_edge; [§2.32.23, page 70]
      PEXVertexCIEEdge            *cie_edge; [§2.32.24, page 70]
      PEXVertexHLSEdge            *hls_edge; [§2.32.25, page 71]
      PEXVertexHSVEdge            *hsv_edge; [§2.32.26, page 71]
      PEXVertexRGBEdge            *rgb_edge; [§2.32.27, page 71]
      PEXVertexRGBAEdge           *rgba_edge; [§2.32.28, page 71]
      PEXVertexRGB8Edge           *rgb8_edge; [§2.32.29, page 71]
      PEXVertexRGBA8Edge          *rgba8_edge; [§2.32.30, page 71]
      PEXVertexRGB16Edge          *rgb16_edge; [§2.32.31, page 71]
      PEXVertexRGBA16Edge         *rgba16_edge; [§2.32.32, page 72]
      PEXVertexNormalEdge         *normal_edge; [§2.32.33, page 72]
      PEXVertexIndexedNormalEdge  *index_normal_edge; [§2.32.34,
page 72]
      PEXVertexCIENormalEdge      *cie_normal_edge; [§2.32.35, page 72]
      PEXVertexHLSNormalEdge      *hls_normal_edge; [§2.32.36, page 72]
      PEXVertexHSVNormalEdge      *hsv_normal_edge; [§2.32.37, page 72]
      PEXVertexRGBNormalEdge      *rgb_normal_edge; [§2.32.38, page 72]
      PEXVertexRGBANormalEdge     *rgba_normal_edge; [§2.32.39,
page 73]
      PEXVertexRGB8NormalEdge     *rgb8_normal_edge; [§2.32.40,
page 73]
      PEXVertexRGBA8NormalEdge    *rgba8_normal_edge; [§2.32.41,
page 73]
      PEXVertexRGB16NormalEdge    *rgb16_normal_edge; [§2.32.42,
page 73]
      PEXVertexRGBA16NormalEdge   *rgba16_normal_edge; [§2.32.43,
page 73]
      PEXPointer                  with_fp_data; [§2.18.1, page 46]
      PEXCoord2D                  *no_data2D; [§2.7.2, page 33]
      PEXVertexIndexed2D          *index2D; [§2.32.44, page 73]
      PEXVertexCIE2D              *cie2D; [§2.32.45, page 73]
      PEXVertexHLS2D              *hls2D; [§2.32.46, page 74]
      PEXVertexHSV2D              *hsv2D; [§2.32.47, page 74]
      PEXVertexRGB2D              *rgb2D; [§2.32.48, page 74]
      PEXVertexRGBA2D             *rgba2D; [§2.32.49, page 74]
      PEXVertexRGB82D             *rgb82D; [§2.32.50, page 74]
      PEXVertexRGBA82D            *rgba82D; [§2.32.51, page 74]
      PEXVertexRGB162D            *rgb162D; [§2.32.52, page 74]
      PEXVertexRGBA162D           *rgba162D; [§2.32.53, page 74]
      PEXVertexNormal2D           *normal2D; [§2.32.54, page 74]
      PEXVertexIndexedNormal2D    *index_normal2D; [§2.32.55, page 75]
      PEXVertexCIENormal2D        *cie_normal2D; [§2.32.56, page 75]
```

```
      PEXVertexHLSNormal2D          *hls_normal2D; [§2.32.57, page 75]
      PEXVertexHSVNormal2D          *hsv_normal2D; [§2.32.58, page 75]
      PEXVertexRGBNormal2D          *rgb_normal2D; [§2.32.59, page 75]
      PEXVertexRGBANormal2D         *rgba_normal2D; [§2.32.60, page 75]
      PEXVertexRGB8Normal2D         *rgb8_normal2D; [§2.32.61, page 75]
      PEXVertexRGBA8Normal2D        *rgba8_normal2D; [§2.32.62, page 76]
      PEXVertexRGB16Normal2D        *rgb16_normal2D; [§2.32.63, page 76]
      PEXVertexRGBA16Normal2D       *rgba16_normal2D; [§2.32.64, page 76]
      PEXVertexEdge2D               *edge2D; [§2.32.65, page 76]
      PEXVertexIndexedEdge2D        *index_edge2D; [§2.32.66, page 76]
      PEXVertexCIEEdge2D            *cie_edge2D; [§2.32.67, page 76]
      PEXVertexHLSEdge2D            *hls_edge2D; [§2.32.68, page 76]
      PEXVertexHSVEdge2D            *hsv_edge2D; [§2.32.69, page 77]
      PEXVertexRGBEdge2D            *rgb_edge2D; [§2.32.70, page 77]
      PEXVertexRGBAEdge2D           *rgba_edge2D; [§2.32.71, page 77]
      PEXVertexRGB8Edge2D           *rgb8_edge2D; [§2.32.72, page 77]
      PEXVertexRGBA8Edge2D          *rgba8_edge2D; [§2.32.73, page 77]
      PEXVertexRGB16Edge2D          *rgb16_edge2D; [§2.32.74, page 77]
      PEXVertexRGBA16Edge2D         *rgba16_edge2D; [§2.32.75, page 77]
      PEXVertexNormalEdge2D         *normal_edge2D; [§2.32.76, page 78]
      PEXVertexIndexedNormalEdge2D  *index_normal_edge2D; [§2.32.77,
page 78]
      PEXVertexCIENormalEdge2D      *cie_normal_edge2D; [§2.32.78,
page 78]
      PEXVertexHLSNormalEdge2D      *hls_normal_edge2D; [§2.32.79,
page 78]
      PEXVertexHSVNormalEdge2D      *hsv_normal_edge2D; [§2.32.80,
page 78]
      PEXVertexRGBNormalEdge2D      *rgb_normal_edge2D; [§2.32.81,
page 78]
      PEXVertexRGBANormalEdge2D     *rgba_normal_edge2D; [§2.32.82,
page 79]
      PEXVertexRGB8NormalEdge2D     *rgb8_normal_edge2D; [§2.32.83,
page 79]
      PEXVertexRGBA8NormalEdge2D    *rgba8_normal_edge2D; [§2.32.84,
page 79]
      PEXVertexRGB16NormalEdge2D    *rgb16_normal_edge2D; [§2.32.85,
page 79]
      PEXVertexRGBA16NormalEdge2D   *rgba16_normal_edge2D; [§2.32.86,
page 79]
      PEXPointer                    with_fp_data2D; [§2.18.1, page 46]
} PEXArrayOfVertex;
```

2.32.88. `PEXListOfVertex` Data Structure

```
typedef struct {
      unsigned long     count;            /* number of vertices */
      PEXArrayOfVertex  vertices; [§2.32.87, page 79]/* pointer to
vertices */
} PEXListOfVertex;
```

2.33. Data Structures for the Visual and Pixmap Depth Selection Utilities

The data structures for visual utility extensions include:

2.33.1. `PEXUtVisualCriteria` Data Structure

```
typedef struct {
      unsigned int      hard_criteria_mask;
      unsigned int      soft_criteria_mask;
      unsigned int      depth;
      unsigned int      min_colors;
      unsigned int      min_red;
      unsigned int      min_green;
      unsigned int      min_blue;
      unsigned int      visual_class;
      unsigned int      standard_colormap_property;
      unsigned int      dynamic_visual;
      unsigned int      double_buffering_capability;
      PEXEnumTypeIndex  color_approx_type;
      unsigned int      z_buffer_depth;
      unsigned int      alpha_buffer_depth;
      unsigned int      accum_buffer_depth;
      unsigned int      accum_buffer_min_red;
      unsigned int      accum_buffer_min_green;
      unsigned int      accum_buffer_min_blue;
} PEXUtVisualCriteria;
```

2.33.2. `PEXUtPixmapDepthCriteria` Data Structure

```
typedef struct {
      unsigned int      hard_criteria_mask;
      unsigned int      soft_criteria_mask;
      unsigned int      depth;
      PEXEnumTypeIndex  color_approx_type;
      unsigned int      z_buffer_depth;
      unsigned int      alpha_buffer_depth;
      unsigned int      accum_buffer_depth;
      unsigned int      accum_buffer_min_red;
      unsigned int      accum_buffer_min_green;
      unsigned int      accum_buffer_min_blue;
} PEXUtPixmapDepthCriteria;
```

2.33.3. Mask Bits for the Hard and Soft Criteria

```
#define PEXUtDepth (1L<<0)
#define PEXUtMinColors (1L<<1)
#define PEXUtMinRed (1L<<2)
#define PEXUtMinGreen (1L<<3)
#define PEXUtMinBlue (1L<<4)
#define PEXUtVisualClass (1L<<5)
#define PEXUtStandardColormapProperty (1L<<6)
#define PEXUtDynamicVisual (1L<<7)
#define PEXUtDBCapability (1L<<8)
#define PEXUtColorApproxType (1L<<9)
#define PEXUtZBufferDepth (1L<<10)
#define PEXUtAlphaBufferDepth (1L << 11)
#define PEXUtAccumBufferDepth (1L << 12)
#define PEXUtAccumBufferMinRed (1L << 13)
#define PEXUtAccumBufferMinGreen (1L << 14)
#define PEXUtAccumBufferMinBlue (1L << 15)
```

2.33.4. Values for the Double_buffering_capability Criterion

```
#define PEXUtDBNone 0
#define PEXUtDBPEX 1
#define PEXUtDBPEXAndX 2
```

2.33.5. Values for the Integer Error Codes Returned by the Various Utilities

```
#define PEXUtSuccess 0
#define PEXUtQualifiedSuccess 1
#define PEXUtAlternativeSuccess 2
#define PEXUtCriteriaFailure -1
#define PEXUtXFailure -2
#define PEXUtPEXFailure -3
#define PEXUtBadAlloc -4
```

Chapter 3

PEXlib Access and Information Functions

This chapter describes the PEXlib functions you need to begin using PEXlib (e.g., **PEXInitialize**) and to obtain information that is specific to the PEX/PEXlib implementation you are using. A few of the more common ones are introduced below.

You must invoke **PEXInitialize** before using most of the PEXlib functions. If **PEXInitialize** succeeds, you may use the information it returns to learn more about the particular PEX/PEXlib implementation, or you may invoke **PEXGetExtensionInfo** to retrieve the same information at a later time.

The PEX specification does not require that PEX implementors completely implement all functions described in the specifications. You use the **PEXGetEnumTypeInfo** function to find out whether the functions you need are present in the implementation you are using. You use the **PEXGetImpDepConstants** to determine the limits involved with some features in the implementation.

PEXMatchRenderingTargets, **PEXQueryColorApproximation**, and **PEXSelectVisual** are beneficial for selecting visuals and for initializing colormaps when you are using the X Window System to create drawables for PEXlib use.

Double Buffering

The **PEXDBAllocateBackBufferName**, **PEXDBFreeBackBufferName**, **PEXDBSwapBuffer**, and **PEXDBTestVisual** functions are useful for creating double-buffered applications. You are encouraged to use these functions instead of older techniques such as the MBX extension, special escape functions, or pixmap double buffering when working with PEX 5.2 implementations. These functions follow the design of the DBE (Double Buffer Extension) which is a required X co-extension for PEX. These PEXlib functions provide nearly all the function and flexibility of the DBE itself. If you need to access DBE functionality not exposed by these PEXlib functions, then it is better to use direct DBE function calls exclusively. PEXlib implementations may or may not use DBE in the implementations of the PEXDB Double Buffering functions, so it may be difficult to predict the interaction between the DBE and the PEXlib implementation.

PEXConvertDoubleFloat - Convert Double Precision Native Floating-Point Numbers to Protocol Floating-Point Format

Synopsis

Status **PEXConvertDoubleFloat**(Display **display,* unsigned int *numfloats*, double **floats*, unsigned int **retsize*, `PEXPointer` **retfloats*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
numfloats	The number of double-precision floating-point numbers to convert.
floats	A pointer to the list of contiguous floating-point numbers to be converted.
retsize	The size, in bytes, of the generated protocol floating-point list returned by this function. This size value is returned by this function in *retsize*.
retfloats	A pointer to the list of protocol floating-point numbers generated by this function. This function allocates space for these returned values and places a pointer to this space in *retfloats* [§2.18.1, page 46].

Returns

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

PEXBadLocalAlloc	Indicates that this function ran out of memory.

Description

Invoke **PEXConvertDoubleFloat** to convert a contiguous list of double precision floating-point numbers from the client program's native floating-point format to the format the Display connection currently uses. This function generates a list of untyped data that you use as a single block of data for the object that requires the converted floating-point data.

The native floating-point format is the format currently being used on the machine on which the PEXlib client application is executing. This function is intended mainly for use with vendor-specific GSE's, GDP's, Escapes, or output commands that contain floating-point data. A standard PEXlib implementation cannot convert the floating-point data while generating protocol if the native and protocol floating-point formats are not the same because the standard implementation is not aware of vendor extensions and the location of the floating-point data within these extensions. If the native floating-point format is different from the protocol floating-point format, then the client program should invoke this function to convert the floating-point data and then use the resulting list to create the GSE, GDP, Escape, or vendor OC.

Specify with the *numfloats* argument the number of floats that this function is to convert.

Specify with the *floats* argument the list of contiguous double-precision floating-point numbers that need to be converted.

The function returns the arguments *retsize* and *retfloats*. You use the **XFree** function to free the storage returned in *retfloats*.

Example:

The following GDP defined by the ACME company takes 6 points and 3 floats as parameters.

```
#define ACMEGDPFOOID 0x80000000

PEXCoord        my_FOO_pts[6] = {....};
double          my_FOO_parms[3] = {1.0, 1.0, 5.0};
unsigned int    retsize;
PEXPointer      retfloats;

PEXConvertDoubleFloat(display, 3, my_FOO_parms, &retsize, &retfloats);
PEXOCCGDP(context, ACMEGDPFOOID, 6, my_FOO_pts, retsize,
            (char*)retfloats);
XFree(retfloats);
```

Errors

None.

See Also

PEXConvertSingleFloat, **PEXGetProtocolFloatFormat**

PEXConvertSingleFloat - Convert Single Precision Native Floating-Point Numbers to Protocol Floating-Point Format

Synopsis

Status **PEXConvertSingleFloat**(Display **display,* unsigned int *numfloats*, float **floats*, unsigned int **retsize*, `PEXPointer` **retfloats*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
numfloats	The number of floating-point numbers to convert.
floats	A pointer to the list of contiguous floating-point numbers to be converted.
retsize	The size, in bytes, of the generated protocol floating-point list returned by this function. This size value is returned by this function in *retsize*.
retfloats	A pointer to the list of protocol floating-point numbers generated by this function. This function allocates space for these returned values and places a pointer to this space in *retfloats* [§2.18.1, page 46].

Returns

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

PEXBadLocalAlloc	Indicates that this function ran out of memory.

Description

Invoke **PEXConvertSingleFloat** to convert a contiguous list of single precision floating-point numbers from the client program's native floating-point format to the format the Display connection currently uses. This function generates a list of untyped data that you use as a single block of data for the object that requires the converted floating-point data.

The native floating-point format is the format currently being used on the machine on which the PEXlib client application is executing. This function is intended mainly for use with vendor-specific GSE's, GDP's, Escapes, or output commands that contain floating-point data. A standard PEXlib implementation cannot convert the floating-point data while generating protocol if the native and protocol floating-point formats are not the same because the standard implementation is not aware of vendor extensions and the location of the floating-point data within these extensions. If the native floating-point format is different from the protocol floating-point format, then the client program should invoke this function to convert the floating-point data and then use the resulting list to create the GSE, GDP, Escape, or vendor OC.

Specify with the *numfloats* argument the number of floats that this function is to convert.

Specify with the *floats* argument the list of contiguous floating-point numbers that need to be converted.

The function returns the parameters *retsize* and *retfloats*. You use the **XFree** function to free the storage returned in *retfloats*.

Example:

The following GDP defined by the ACME company takes 6 points and 3 floats as parameters.

```
#define ACMEGDPFOOID 0x80000000

PEXCoord        my_FOO_pts[6] = {....};
float           my_FOO_parms[3] = {1.0, 1.0, 5.0};
unsigned int    retsize;
PEXPointer      retfloats;

PEXConvertSingleFloat(display, 3, my_FOO_parms, &retsize, &retfloats);
PEXOCCGDP(context, ACMEGDPFOOID, 6, my_FOO_pts, retsize,
            (char*)retfloats);
XFree(retfloats);
```

Errors

None.

See Also

PEXConvertDoubleFloat, **PEXGetProtocolFloatFormat**

PEXDBAllocateBackBufferName - Allocate a Name for a Window’s Back Buffer

Synopsis

Drawable **PEXDBAllocateBackBufferName**(Display **display*, Window *window*, int *hint*)

Arguments

display	A pointer to a display structure returned by a successful XOpenDisplay call.
window	The ID of the window for which to allocate the back buffer name.
hint	One of **PEXDBSwapUndefined**, **PEXDBSwapBackground**, **PEXDBSwapUntouched**, **PEXDBSwapCopied**. See “PEXDBSwapBuffer - Swap a Window with its Back Buffer” on page 92 for a description of the swap actions.

Returns

A Drawable ID that references the back buffer for the specified *window*.

Description

Invoke **PEXDBAllocateBackBufferName** to request an ID for the back buffer associated with the specified *window*. If the specified *window* does not have a back buffer already associated with it, the implementation allocates the back buffer and returns a name associated with it. If the *window* already has a back buffer, the implementation does not allocate a new back buffer, but instead returns a unique ID that references the same back buffer already associated with the specified *window*. It is impossible for the client to allocate more than one back buffer for a given window.

Supply one of the values for *hint* to indicate the swap action that you are most likely to use for subsequent swap requests.

Errors

BadAlloc	The implementation failed to allocate memory to satisfy request.
BadMatch	The specified *window* is not capable of double buffering.
BadValue	The specified *hint* is invalid.
BadWindow	The specified *window* is invalid.

See Also

PEXDBFreeBackBufferName

PEXDBFreeBackBufferName - Free a Name for a Window's Back Buffer

Synopsis

void **PEXDBFreeBackBufferName**(Display **display*, Drawable *back_buffer*)

Arguments

display	A pointer to a display structure returned by a successful XOpenDisplay call.
back_buffer	The ID (name) of the back buffer to free.

Returns

None.

Description

Invoke **PEXDBFreeBackBufferName** to remove the association of the name *back_buffer* to the back buffer belonging to a window capable of double buffering. If this is the last back buffername to be freed for its window, the implementation is free to deallocate resources associated with the back buffer, at its discretion. The name *back_buffer* is no longer valid for use after invoking this function.

Errors

BadMatch	The specified *back_buffer* does not refer to a back buffer associated with a window.

See Also

PEXDBAllocateBackBufferName

PEXDBSwapBuffer - Swap a Window with its Back Buffer

Synopsis

void **PEXDBSwapBuffer** (Display **display*, PEXRenderer *renderer*, int *swap_action*, unsigned long *clear_mask*)

Arguments

display	A pointer to a display structure returned by a successful XOpenDisplay call.
renderer	The ID of a renderer bound to the window to swap.
swap_action	One of **PEXDBSwapUndefined**, **PEXDBSwapBackground**, **PEXDBSwapUntouched**, **PEXDBSwapCopied**.
clear_mask	A single word bitmask containing zero or the logical OR of any of the following: **PEXClearRendererImage**, **PEXClearRendererZ**, **PEXClearRendererAlpha**.

Returns

None.

Description

Invoke **PEXDBSwapBuffer** to swap the back and front buffers for the window associated with the specified *renderer*. The swapping action does not modify the ID's associated with the window and the back buffer, so the application may continue to reference the "front buffer" and the "back buffer" with the same ID's it used to refer to the window and back buffer before the swap request.

After the swap, the implementation first performs the specified *swap_action* on the back buffer as follows:

PEXDBSwapUndefined - the implementation leaves the contents of the back buffer in the state most convenient for the particular implementation. The client cannot count on any particular operation on the back buffer for this action.

PEXDBSwapBackground - the implementation tiles the unobscured regions of the back buffer with the window's background, using the window's attributes as specified by the X protocol.

PEXDBSwapUntouched - the implementation leaves the unobscured regions of the back buffer unmodified.

PEXDBSwapCopied - the implementation makes the contents of the back buffer the same as the contents of the window.

The implementation then performs clear functions using the PEX renderer and PEX renderer attributes after the *swap_action* is complete, according to the settings of the flag in *clear_mask*. The implementation may use the renderer's background color, background color's alpha (1.0 if not present), Z clear value, or clip list attributes during the clear action, again depending on the *clear_mask*.

Note that the implementation processes the *swap_action* first, followed by the clear action specified by *clear_mask*. When the renderer's clip list is empty, a renderer clear action will "wipe-out" the result of the *swap_action*. In this case, the implementation may optimize the swap, *swap_action*, and clear action into a single swap and clear, using the renderer's background color.

As another example, if the renderer has a non-empty clip list and a clear image is called for by the client, then the implementation cannot optimize away the *swap_action* because the results of the *swap_action* are visible outside of the renderer's clip list. If the *swap_action* is **PEXDBSwapBackground**, then the X window's background tiling will be visible outside the ranges defined by the clip list. The implementation fills the regions within the clip list with the renderer's background color.

Errors

BadAlloc	The implementation failed to allocate memory to satisfy request.
BadMatch	The specified renderer's window is not double buffered.
BadPEXRendererState	The renderer's state is idle.
BadValue	The specified *swap_action* is invalid.

See Also

PEXDBAllocateBackBufferName

PEXDBTestVisual - Test a Visual for Double Buffering Capability

Synopsis

int **PEXDBTestVisual** (Display **display*, Drawable *drawable*, VisualID *visualid*)

Arguments

display	A pointer to a display structure returned by a successful XOpenDisplay call.
drawable	The ID of a drawable used to identify the screen for which the visual is tested.
visualid	The ID of the visual to test.

Returns

True if the specified visual supports double buffering. **False** if the specified visual does not support double buffering.

Description

Invoke **PEXDBTestVisual** to determine if a window created with the given visual supports double buffering through the use of the PEXlib double buffering functions described in this chapter.

You use this function to “pre-screen” a visual prior to specifying it in a **PEXMatchRenderingTargets** invocation, or you use this function to test visuals returned by **PEXMatchRenderingTargets** to determine if they can be used for creating windows that are double buffered.

Errors

BadDrawable	The specified drawable is invalid.

See Also

PEXDBAllocateBackBufferName, **PEXMatchRenderingTargets**, **PEXSelectVisual**

PEXFreeAncillaryBufferInfo - Free Ancillary BufferInformation (5.2)

Synopsis

void **PEXFreeAncillaryBufferInfo**(Display **display*, unsigned long *num_visuals*, unsigned long *num_depths*, `PEXAncillaryBufferInfo` ***visual_info*, `PEXAncillaryBufferInfo` ***depth_info*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
num_visuals	The number of visuals.
num_depths	The number of depths.
visual_info	The address of an array of pointers to `PEXAncillaryBufferInfo` [§2.4.1, page 29].
depth_info	The address of an array of pointers to `PEXAncillaryBufferInfo` [§2.4.1, page 29].

Returns

None.

Description

Invoke **PEXFreeAncillaryBufferInfo** to free the storage allocated by **PEXGetAncillaryBufferInfo**.

Errors

None

See Also

PEXGetAncillaryBufferInfo

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* argument in the **PEXGetEnumTypeInfo** function.
enum_info	An array of enumerated type descriptors [§2.9.1, page 34]. This corresponds to the *enum_info_return* argument of the **PEXGetEnumTypeInfo** function.

Returns

None.

Description

Invoke **PEXFreeEnumInfo** to free memory allocated by PEXlib for the return value and return argument in the **PEXGetEnumTypeInfo** function.

Errors

None.

See Also

PEXGetEnumTypeInfo

PEXGetAncillaryBufferInfo - Get Ancillary Buffer Information (5.2)

Synopsis

Status **PEXGetAncillaryBufferInfo**(Display **display*, Drawable *drawable*, unsigned long *num_visuals*, unsigned long *num_depths*, VisualID **visual_ids*, unsigned long **depths*, `PEXAncillaryBufferInfo` ****visual_info*, `PEXAncillaryBufferInfo` ****depth_info*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable used to identify the screen for which the information is to be obtained.
num_visuals	The number of visuals.
num_depths	The number of pixmap depths.
visual_ids	The list of visual IDs about which you want information returned.
depths	The list of pixmap depths about which you want information returned.
visual_info	The address of a pointer to an array of pointers to `PEXAncillaryBufferInfo`. The function stores the address of the array in the supplied pointer [§2.4.1, page 29].
depth_info	The address of a pointer to an array of pointers to `PEXAncillaryBufferInfo`. The function stores the address of the array in the supplied pointer [§2.4.1, page 29].

Returns

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

PEXBadLocalAlloc	Indicates that this function ran out of memory.

Description

Invoke **PEXGetAncillaryBufferInfo** to retrieve information about Z-buffers, alpha buffers, and accumulation buffers associated with visuals or pixmap depths.

This function returns the address of an array of pointers to the `PEXAncillaryBufferInfo` data structure. You invoke the function and use the returned data as follows:

```
PEXAncillaryBufferInfo **info;
unsigned long depths[]={8,24};

PEXGetAncillaryBufferInfo(display, win, 0, 2, NULL, depths, NULL, &info);
accum_depth8 = info[0]->accum_depth;
accum_depth24 = info[1]->accum_depth;
PEXFreeAncillaryBufferInfo(display, 0, 2, NULL, info);
```

PEXlib allocates memory for the returned array of pointers and PEXAncillaryBufferInfo structures. Invoke the **PEXFreeAncillaryBufferInfo** function to deallocate this memory. (See“Freeing Memory Allocated by PEX-lib functions” on page 22.)

Errors

BadDrawable	The specified drawable resource identifier is invalid.
BadValue	One of the specified *visual_ids* is invalid.

See Also

PEXFreeAncillaryBufferInfo

PEXGetEnumTypeInfo - Get Enumerated Type Information (5.0)

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	The list of enumerated types about which you want information returned. (See the Description for valid values.)
item_mask	A mask indicating the data you want 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 [§2.9.1, page 34].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetEnumTypeInfo** to get the supported values for each enumerated type.

For each specified enumerated type you request, this function returns a descriptor 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. These fields contain values valid for all drawables having the same root window, depth and visual ID as the specified drawable.

Use the *item_mask* argument to specify the type of data you want returned in each enumerated type descriptor. PEXlib defines the following *item_mask* values:

PEXETCounts - the function returns only the counts, not the descriptor values.
PEXETIndex - the function returns the supported index values in addition to the counts.
PEXETMnemonic - the function returns the mnemonic strings describing the supported values in addition to the counts. The mnemonic strings are returned in the ISO-LATIN1 character encoding.
PEXETAll - the function returns both the supported index values and the mnemonic strings in addition to the counts.

PEXlib allocates memory for the returned array of counts and for the return value array of enumerated type descriptors. Invoke the **PEXFreeEnumInfo** function to deallocate this memory. (See"Freeing Memory Allocated by PEXlib functions" on page 22.)

The following are the standard enumerated type index, mnemonic pairs which this function may return for each standard enumerated type.

Note: A convenience function is provided for determining if an alpha blend function is supported by the implementation, given a source blend factor and a destination blend factor pair. See "PEXQueryAlphaBlend - Query Support for Alpha Blend Factors (5.2)" on page 270. The list of 42 enum type values listed below correspond directly to the entries in Table 8, "Defined Alpha Blend Functions," on page 266.

PEXETAlphaBlendFunction (entire enum type new for 5.2)

PEXAlphaBlendNone	**PEXETMAlphaBlendNone**
PEXAlphaBlendSimple	**PEXETMAlphaBlendSimple**
PEXAlphaBlendFunction2	**PEXETMAlphaBlendFunction2**
PEXAlphaBlendFunction3	**PEXETMAlphaBlendFunction3**
PEXAlphaBlendFunction4	**PEXETMAlphaBlendFunction4**
PEXAlphaBlendFunction5	**PEXETMAlphaBlendFunction5**
PEXAlphaBlendFunction6	**PEXETMAlphaBlendFunction6**
PEXAlphaBlendFunction7	**PEXETMAlphaBlendFunction7**
PEXAlphaBlendFunction8	**PEXETMAlphaBlendFunction8**
PEXAlphaBlendFunction9	**PEXETMAlphaBlendFunction9**
PEXAlphaBlendFunction10	**PEXETMAlphaBlendFunction10**
PEXAlphaBlendFunction11	**PEXETMAlphaBlendFunction11**
PEXAlphaBlendFunction12	**PEXETMAlphaBlendFunction12**
PEXAlphaBlendFunction13	**PEXETMAlphaBlendFunction13**
PEXAlphaBlendFunction14	**PEXETMAlphaBlendFunction14**
PEXAlphaBlendFunction15	**PEXETMAlphaBlendFunction15**
PEXAlphaBlendFunction16	**PEXETMAlphaBlendFunction16**
PEXAlphaBlendFunction17	**PEXETMAlphaBlendFunction17**
PEXAlphaBlendFunction18	**PEXETMAlphaBlendFunction18**
PEXAlphaBlendFunction19	**PEXETMAlphaBlendFunction19**
PEXAlphaBlendFunction20	**PEXETMAlphaBlendFunction20**
PEXAlphaBlendFunction21	**PEXETMAlphaBlendFunction21**
PEXAlphaBlendFunction22	**PEXETMAlphaBlendFunction22**
PEXAlphaBlendFunction23	**PEXETMAlphaBlendFunction23**
PEXAlphaBlendFunction24	**PEXETMAlphaBlendFunction24**
PEXAlphaBlendFunction25	**PEXETMAlphaBlendFunction25**
PEXAlphaBlendFunction26	**PEXETMAlphaBlendFunction26**
PEXAlphaBlendFunction27	**PEXETMAlphaBlendFunction27**
PEXAlphaBlendFunction28	**PEXETMAlphaBlendFunction28**
PEXAlphaBlendFunction29	**PEXETMAlphaBlendFunction29**
PEXAlphaBlendFunction30	**PEXETMAlphaBlendFunction30**
PEXAlphaBlendFunction31	**PEXETMAlphaBlendFunction31**
PEXAlphaBlendFunction32	**PEXETMAlphaBlendFunction32**
PEXAlphaBlendFunction33	**PEXETMAlphaBlendFunction33**
PEXAlphaBlendFunction34	**PEXETMAlphaBlendFunction34**
PEXAlphaBlendFunction35	**PEXETMAlphaBlendFunction35**
PEXAlphaBlendFunction36	**PEXETMAlphaBlendFunction36**
PEXAlphaBlendFunction37	**PEXETMAlphaBlendFunction37**
PEXAlphaBlendFunction38	**PEXETMAlphaBlendFunction38**
PEXAlphaBlendFunction39	**PEXETMAlphaBlendFunction39**
PEXAlphaBlendFunction40	**PEXETMAlphaBlendFunction40**
PEXAlphaBlendFunction41	**PEXETMAlphaBlendFunction41**

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**
PEXColorTypeRGBA	**PEXETMColorTypeRGBA** (new for 5.2)
PEXColorTypeRGBA8	**PEXETMColorTypeRGBA8** (new for 5.2)
PEXColorTypeRGBA16	**PEXETMColorTypeRGBA16** (new for 5.2)

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**

PEXETDataMapMethod (entire enum type new for 5.2)

PEXDataMapNone	**PEXETMDataMapNone**
PEXDataMapColor	**PEXETMDataMapColor**

PEXETDisplayUpdateMode

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

PEXETEchoMethod (entire enum type new for 5.2)

PEXEchoDestructiveColor	**PEXETMEchoDestructiveColor**
PEXEchoNonDestructiveColor	**PEXETMEchoNonDestructiveColor**
PEXEchoXORColor	**PEXETMEchoXORColor**

PEXETEscape (entire enum type new for 5.1)

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

PEXHatchStyle45Degrees	**PEXETMHatchStyle45Degrees** (new for 5.2)
PEXHatchStyle135Degrees	**PEXETMHatchStyle135Degrees** (new for 5.2)

PEXETHighlightMethod (entire enum type new for 5.2)

PEXHighlightColor	**PEXETMHighlightColor**

PEXETHLHSRIdentifier (entire enum type new for 5.2)

PEXHLHSRIDDisable	**PEXETMHLHSRIDDisable**
PEXHLHSRIDEnable	**PEXETMHLHSRIDEnable**
PEXHLHSRIDEnableTestOnly	**PEXETMHLHSRIDEnableTestOnly**

PEXETHLHSRMode

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

PEXETInteriorStyle

PEXInteriorStyleHollow	**PEXETMInteriorStyleHollow**
PEXInteriorStyleSolid	**PEXETMInteriorStyleSolid**
PEXInteriorStylePattern	**PEXETMInteriorStylePattern**
PEXInteriorStyleHatch	**PEXETMInteriorStyleHatch**
PEXInteriorStyleEmpty	**PEXETMInteriorStyleEmpty**
PEXInteriorStyleTexture	**PEXETMInteriorStyleTexture** (new for 5.2)

PEXETLightType

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

PEXETLineCapStyle (entire enum type new for 5.2)

PEXLineCapButt	**PEXETMLineCapButt**
PEXLineCapRound	**PEXETMLineCapRound**
PEXLineCapProject	**PEXETMLineCapProject**

PEXETLineJoinStyle (entire enum type new for 5.2)

PEXLineJoinImpDep	implementation-dependent string
PEXLineJoinRound	**PEXETMLineJoinRound**
PEXLineJoinMiter	**PEXETMLineJoinMiter**

PEXLineJoinBevel	**PEXETMLineJoinBevel**

PEXETLineType

PEXLineTypeSolid	**PEXETMLineTypeSolid**
PEXLineTypeDashed	**PEXETMLineTypeDashed**
PEXLineTypeDotted	**PEXETMLineTypeDotted**
PEXLineTypeDashDot	**PEXETMLineTypeDashDot**
PEXLineTypeCenterLine	**PEXETMLineTypeCenterLine** (new for 5.2)
PEXLineTypePhantom	**PEXETMLineTypePhantom** (new for 5.2)

PEXETMarkerGlyphType (entire enum type new for 5.2)

PEXMarkerGlyphStroke	**PEXETMMarkerGlyphStroke**

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 (entire enum type new for 5.1)

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

PEXETPickDeviceType

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

PEXETPickOneMethod (entire enum type new for 5.1)

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

PEXETPolylineInterpMethod

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

PEXETPrimitiveAABlendFunction (entire enum type new for 5.2)

PEXPrimitiveAABlendImpDep	**PEXETMPrimitiveAABlendImpDep**
PEXPrimitiveAABlendSimpleAlpha	**PEXETMPrimitiveAABlendSimpleAlpha**

PEXETPromptEchoType

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

PEXETReflectionModel

The first four index/mnemonic pairs have the same values and meaning as the last four. The first four are compatible with the PEXlib 5.1 specification, but are also available in PEXlib 5.2 for compatibility with existing PEXlib 5.1 applications. Use the last four in PEXlib 5.2 and later applications.

PEXReflectionNone	**PEXETMReflectionNone**
PEXReflectionAmbient	**PEXETMReflectionAmbient**
PEXReflectionDiffuse	**PEXETMReflectionDiffuse**
PEXReflectionSpecular	**PEXETMReflectionSpecular**
PEXRMNone	**PEXETMRMNone** (new for 5.2)
PEXRMAmbient	**PEXETMRMAmbient** (new for 5.2)
PEXRMAmbientDiffuse	**PEXETMRMAmbientDiffuse** (new for 5.2)
PEXRMAmbientDiffuseSpecular	**PEXETMRMAmbientDiffuseSpecular** (new for 5.2)

PEXETRenderingColorModel

PEXRenderingColorModelImpDep	implementation-dependent string
PEXRenderingColorModelRGB	**PEXETMRenderingColorModelRGB**
PEXRenderingColorModelCIE	**PEXETMRenderingColorModelCIE**

PEXRenderingColorModelHSV	**PEXETMRenderingColorModelHSV**
PEXRenderingColorModelHLS	**PEXETMRenderingColorModelHLS**

PEXETSupportedTexelResources (entire enum type new for 5.2)

PEXTexelResourcesWindow	**PEXETMTexelResourcesWindow**
PEXTexelResourcesPixmap	**PEXETMTexelResourcesPixmap**
PEXTexelResourcesBuffer	**PEXETMTexelResourcesBuffer**
PEXTexelResourcesPhotomap	**PEXETMTexelResourcesPhotomap**

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**

PEXETTMBoundaryClampMethod (entire enum type new for 5.2)

PEXTMBoundaryClampExplicitDiscontinue	**PEXETMTMBoundaryClampExplicitDiscontinue**
PEXTMBoundaryClampExplicitColor	**PEXETMTMBoundaryClampExplicitColor**

PEXETTMBoundaryCondition (entire enum type new for 5.2)

PEXTMBoundaryClampExplicit	**PEXETMTMBoundaryClampExplicit**
PEXTMBoundaryClampBorder	**PEXETMTMBoundaryClampBorder**

PEXTMBoundaryWrap	**PEXETMTMBoundaryWrap**
PEXTMBoundaryMirror	**PEXETMTMBoundaryMirror**

PEXETTMCompositionMethod (entire enum type new for 5.2)

PEXTMCompReplace	**PEXETMTMCompReplace**
PEXTMCompModulate	**PEXETMTMCompModulate**
PEXTMCompBlendEnvColor	**PEXETMTMCompBlendEnvColor**
PEXTMCompDecal	**PEXETMTMCompDecal**
PEXTMCompDecalBackground	**PEXETMTMCompDecalBackground**

PEXETTMDimension (entire enum type new for 5.2)

PEXTMDimension1D	**PEXETMTMDimension1D**
PEXTMDimension2D	**PEXETMTMDimension2D**
PEXTMDimension3D	**PEXETMTMDimension3D**

PEXETTMParameterizationMethod (entire enum type new for 5.2)

PEXTMParamVertexMCCoord	**PEXETMTMParamVertexMCCoord**
PEXTMParamVertexWCCoord	**PEXETMTMParamVertexWCCoord**
PEXTMParamVertexMCNormal	**PEXETMTMParamVertexMCNormal**
PEXTMParamVertexWCNormal	**PEXETMTMParamVertexWCNormal**
PEXTMParamNURBUV	**PEXETMTMParamNURBUV**
PEXTMParamFloatData	**PEXETMTMParamFloatData**
PEXTMParamReflectionSphereVRC	**PEXETMTMParamReflectionSphereVRC**
PEXTMParamReflectionSphereWC	**PEXETMTMParamReflectionSphereWC**

PEXETTMPerspectiveCorrection (entire enum type new for 5.2)

PEXTMPerspectiveCorrectionNone	**PEXETMTMPerspectiveCorrectionNone**
PEXTMPerspectiveCorrectionVertex	**PEXETMTMPerspectiveCorrectionVertex**
PEXTMPerspectiveCorrectionPixel	**PEXETMTMPerspectiveCorrectionPixel**

PEXETTMRenderingControl (entire enum type new for 5.2)

PEXTMRenderingControlPreLight	**PEXETMTMRenderingControlPreLight**
PEXTMRenderingControlPostLight	**PEXETMTMRenderingControlPostLight**

PEXETTMResourceHints (entire enum type new for 5.2)

PEXTMResourceHintsNone	**PEXETMResourceHintsNone**
PEXTMResourceHintsSpeed	**PEXETMResourceHintsSpeed**
PEXTMResourceHintsSpace	**PEXETMResourceHintsSpace**

PEXETTMSampleFrequency (entire enum type new for 5.2)

PEXTMSampleFrequencyPixel	**PEXETMTMSampleFrequencyPixel**
PEXTMSampleFrequencyInterpDep	**PEXETMTMSampleFrequencyInterpDep**

PEXETTMSampleMethod (entire enum type new for 5.2)

PEXTMSampleSingleBase	**PEXETMTMSampleSingleBase**
PEXTMSampleWeightedBase	**PEXETMTMSampleWeightedBase**
PEXTMSampleSingleInMipmap	**PEXETMTMSampleSingleInMipmap**
PEXTMSampleWeightedInMipmap	**PEXETMTMSampleWeightedInMipmap**
PEXTMSampleSingleBetweenMipmaps	**PEXETMTMSampleSingleBetweenMipmaps**
PEXTMSampleWeightedBetweenMipmaps	**PEXETMTMSampleWeightedBetweenMipmaps**

PEXETTMTexelType (entire enum type new for 5.2)

PEXTMTexelTypeLum	**PEXETMTMTexelTypeLum**
PEXTMTexelTypeLum8	**PEXETMTMTexelTypeLum8**
PEXTMTexelTypeLum16	**PEXETMTMTexelTypeLum16**
PEXTMTexelTypeLumA	**PEXETMTMTexelTypeLumA**
PEXTMTexelTypeLumA8	**PEXETMTMTexelTypeLumA8**
PEXTMTexelTypeLumA16	**PEXETMTMTexelTypeLumA16**
PEXTMTexelTypeRGB	**PEXETMTMTexelTypeRGB**
PEXTMTexelTypeRGB8	**PEXETMTMTexelTypeRGB8**
PEXTMTexelTypeRGB16	**PEXETMTMTexelTypeRGB16**
PEXTMTexelTypeRGBA	**PEXETMTMTexelTypeRGBA**
PEXTMTexelTypeRGBA8	**PEXETMTMTexelTypeRGBA8**
PEXTMTexelTypeRGBA16	**PEXETMTMTexelTypeRGBA16**

PEXETTransparencyMethod (entire enum type new for 5.2)

PEXTransparencyNone	**PEXETMTransparencyNone**
PEXTransparencyScreenDoor	**PEXETMTransparencyScreenDoor**
PEXTransparencyAlphaBlend	**PEXETMTransparencyAlphaBlend**
PEXTransparencyTwoPass	**PEXETMTransparencyTwoPass**
PEXTransparencyMultipass	**PEXETMTransparencyMultipass**

PEXETTrimCurveApproxMethod

PEXApproxImpDep	implementation-dependent string
PEXApproxConstantBetweenKnots	**PEXETMApproxConstantBetweenKnots**

PEXETZBufferTest (entire enum type new for 5.2)

PEXZBufferTestLessThan	**PEXETMZBufferTestLessThan**
PEXZBufferTestLessThanOrEqual	**PEXETMZBufferTestLessThanOrEqual**
PEXZBufferTestGreaterThan	**PEXETMZBufferTestGreaterThan**

PEXZBufferTestGreaterThanOrEqual	**PEXETMZBufferTestGreaterThanOrEqual**
PEXZBufferTestEqual	**PEXETMZBufferTestEqual**
PEXZBufferTestNotEqual	**PEXETMZBufferTestNotEqual**
PEXZBufferTestPass	**PEXETMZBufferTestPass**

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 (5.0)

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 [§2.11.1, page 35].

Description

Invoke **PEXGetExtensionInfo** to get the extension information from a PEX implementation.

This function returns the following information:

Major version number - indicates the major version of the PEX protocol available to the connection.
Minor version number - indicates the minor version of the PEX protocol available to the connection.
Release field - contains a value that represents the vendors' own product's versioning schemes.
Vendor name - contains a string of ISO-LATIN1 characters that identify the vendor of the PEX implementation and its value is implementation-dependent.
Subset information - contains information about the PEX implementation:

> **PEXCompleteImplementation** - the extension is a full PEX implementation and supports all PEX requests.
> **PEXImmediateMode** - the extension supports only the "immediate rendering" subset of PEX.
> **PEXStructureMode** - the extension supports only the "structure rendering" subset of PEX.
> Both **PEXImmediateMode & PEXStructureMode** - the extension supports both the "immediate rendering" and the "structure rendering" subsets of PEX.
> **PEXWorkstationOnly** - the extension supports only the "PHIGS workstation" subset of PEX.

The most significant 16 bits of the *subset_info* field are implementation-dependent. Do *not* include these 16 bits in any test involving the least significant 16 bits.

See the *major_opcode*, *first_event*, and *first_error* fields for the values the X server selected upon initializing the PEX extension. These fields can change between X server start-ups and may depend on other extensions loaded.

Your application must not modify or free the memory returned by this function.

Errors

None

See Also

PEXInitialize

PEXGetImpDepConstants - Get Implementation-Dependent Constants (5.0)

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 that you want returned. (See the Description for valid values.)
constants_return	Returns an array of implementation-dependent constants [§2.15.1, page 39].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetImpDepConstants** to get one or more implementation-dependent constants supported.

For each value requested, this function returns a single integer or floating-point value. Values are returned in order, with one return value in *constants_return* for each item you specify in *names*. This function returns implementation-dependent constants based on the values your application uses for a drawable having the same root window, depth and visual ID as the drawable you specify.

PEXlib allocates memory for the returned constants. Invoke **XFree** to deallocate the memory.

PEXlib defines the following standard implementation-dependent constants:

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 the implementation can compose pixels using shifts and adds.

PEXIDConcentrationClampSupported (integer) - **True** if the implementation applies the concentration clamping region in the reflection calculation involving a light of type **PEXLightWCSpot**. **False** if the implementation does not. (new for 5.2)

PEXIDDitheringControllable (integer) - **True** if **PEXIDDitheringSupported** is **True** and you can turn dithering on and off with the dither field of a color approximation table. **False** if **PEXIDDitheringSupported** is **False**, or if **PEXIDDitheringSupported** is **True** and you cannot disable dithering. (new for 5.2)

PEXIDDitheringSupported (integer) - **True** if the implementation uses the dithering hint to control dithering in the color approximation lookup tables. **False**, if the implementation does not use the dithering hint in color approximation lookup tables.

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

PEXIDEyepointUsed (integer) - **True** if the implementation uses the eyepoint defined in the view table. **False** if the implementation ignores the eyepoint defined in the view table and calculates an implicit eyepoint itself. (new for 5.2)

PEXIDMaxEdgeWidth (integer) - The width (in pixels) of the widest edge your implementation supports.

PEXIDMaxFastTMSize (integer) - The maximum size of any dimension of the base level of a texture map which your implementation can optimize. The implementation might support larger texture maps but the result is implementation-dependent (e.g., may be scaled down to fit the maximum size or may not take advantage of implementation optimization). A value of zero indicates all sizes are equally optimized. (new for 5.2)

PEXIDMaxHitsEventSupported (integer) - **True** if the implementation supports the **PEXMaxHitsReached** event. **False**, if the implementation does not support the event. (new for 5.1)

PEXIDMaxLineWidth (integer) - The width (in pixels) of the widest line or curve that your implementation supports.

PEXIDMaxMarkerSize (integer) - The largest dimension (either height or width, in pixels) of the largest marker that your implementation can draw. This maximum is exclusive of the marker type **PEXMarkerDot** which the implementation always draws as the smallest point able to be displayed.

PEXIDMaxMarkerVertices (integer) - The maximum number of vertices which your implementation can use to define a marker of marker glyph type **PEXMarkerGlyphStroke** in the marker glyph table. (new for 5.2)

PEXIDMaxModelClipPlanes (integer) - The maximum number of modeling clipping planes your implementation can define.

PEXIDMaxNameSetNames (integer) - The maximum number of names your implementation can place in a name set.

PEXIDMaxNonAmbientLights (integer) - The maximum number of non-ambient light sources that your implementation can enable at one time.

PEXIDMaxNumPickResults (integer) - The maximum number of pick results that your implementation can accumulate. (new for 5.2)

PEXIDMaxNURBOrder (integer) - The maximum non-uniform rational B-spline order your implementation can define.

PEXIDMaxTextureMaps (integer) - The maximum number of front- or back-facing texture maps which your implementation can have active for a single primitive. For example, a value of one indicates that your implementation can support one front-facing and one back-facing texture map for a single primitive. A value of zero indicates any number of texture maps can be applied to a single primitive. (new for 5.2)

PEXIDMaxTrimCurveOrder (integer) - The maximum order for trim curves that your implementation can support.

PEXIDMinEdgeWidth (integer) - The width (in pixels) of the thinnest edge that your implementation supports.

PEXIDMinLineWidth (integer) - The width (in pixels) of the thinnest line or curve that your implementation supports.

PEXIDMinMarkerSize (integer) - The largest dimension (either height or width, in pixels) of the smallest marker that your implementation can draw. This minimum is exclusive of the marker type **PEXMarkerDot** which the implementation always draws as the smallest point able to be displayed.

PEXIDNominalEdgeWidth (integer) - The width (in pixels) of the "standard" edge.

PEXIDNominalLineWidth (integer) - The width (in pixels) of the "standard" line or curve.

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

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

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

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

PEXIDPixelColorSupported (integer) - **True** if the renderer's pixel color mode attribute behaves as specified when its value is **PEXOn**. **False** if the pixel color mode has no effect. (new for 5.2)

PEXIDPowerOfTwoTMSizesRequired (integer) - **True** if the implementation requires that the dimensions, in all directions, of all texel arrays defining a texture map must be powers of two. **False** if it does not. (new for 5.2)

PEXIDPrimitiveAASupported (integer) - This is a bitmask which has bit definitions that are located in "PEXOCCSetPrimitiveAntialiasing - Set Primitive Antialiasing (5.2)" on page 376. Each defined bit corresponding to a type of supported primitive antialiasing is set to 1. The defined bits set to 0 correspond to types of primitive antialiasing which are not supported. All undefined bits are set to 0. (new for 5.2)

PEXIDQuadraticAttenSupported (integer) - **True** if the implementation applies the quadratic attenuation coefficient in the reflectance calculation involving a light of type **PEXLightWCPoint** or **PEXLightWCSpot**. **False** if the implementation ignores the coefficient. (new for 5.2)

PEXIDSamplingHintsUsed (integer) - **True** if the implementation uses the sampling hint of the texture map lookup table to make adjustments when sampling a texture map. **False** if the implementation does not use the dithering hint. (new for 5.2)

PEXIDSquareTMRequired (integer) - **True** the implementation requires that each level of a texture map have equally sized dimensions. **False** if it does not. (new for 5.2)

PEXIDTransparencySupported (integer) - **True** if the implementation applies the transmission coefficient in the reflectance calculations. **False** if the implementation ignores the transmission coefficient.

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.

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

PEXGetProtocolFloatFormat - Return Float Format used on Specified Display Connection (5.0)

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

Invoke **PEXGetProtocolFloatFormat** to get 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

Invoke **PEXGetSizeOCs** to get information about the size of the protocol for the output commands.

If the implementation does not support the specified floating-point format, then the implementation issues an unsuccessful return value of zero.

Data Structures

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

Errors

None.

See Also

PEXInitialize - Initialize PEXlib display connection (5.0)

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) [§2.11.1, page 35].
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	The PEX extension does not exist.
PEXBadProtocolVersion	The PEX extension does not support a compatible protocol version, where the major protocol version numbers are equal.
PEXBadFloatConversion	The PEX extension does not support a protocol floating-point format compatible with PEXlib's native format or a format to which PEXlib supports conversion.
PEXBadLocalAlloc	PEXlib client-side allocation failed.

Description

Invoke **PEXInitialize** to initialize PEXlib for the specified display.

You can invoke this function multiple times; subsequent invocations result in the same return value as the first invocation.

If **PEXInitialize** is successful (return value is zero), or if the implementation issues a return value of *PEXBadProtocolVersion*, then this function returns a pointer to the extension information in *info_return*. Otherwise, this function returns a NULL pointer in *info_return*. Your application must not modify or free the extension information. See the function "PEXGetExtensionInfo - Get Extension Information (5.0)" on page 110 for an explanation of the *info_return* argument.

The *error_string* argument specifies a pointer to memory allocated by the application. PEXlib defines the constant **PEXErrorStringLength** as a guideline for the size your application should allocate for the error string. If no failure occurs, then PEXlib does not change the memory addressed by the *error_string* argument. If a failure does occur, then PEXlib copies an error string giving further information about the failure into this memory (up to the maximum specified by the *length* argument).

The actual string returned is implementation-dependent, and PEXlib provides it for convenience in printing an error message for your application's end-user.

Errors

None.

See Also

PEXGetEnumTypeInfo, PEXGetExtensionInfo, PEXGetImpDepConstants, PEXMatchRenderingTargets

PEXMakeColormap - Create a Colormap and Initialize it for PEX Color Approximation

Synopsis

int **PEXMakeColormap**(Display **display*, `XVisualInfo` **vis_info*, `PEXColorApproxEntry` **capx_info*, Colormap **colormap_id_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
vis_info	A pointer to a valid `XVisualInfo` structure that describes the Visual with which the Colormap will be created.
capx_info	A pointer to a valid `PEXColorApproxEntry` structure, that describes the color approximation setup for which the Colormap is to be initialized. [§2.16.3, page 39]
colormap_id_return	A pointer to a Colormap identifier. This function returns the resource ID of the created Colormap at this location.

Returns

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

PEXBadLocalAlloc	Indicates that this function ran out of memory.

Description

Invoke **PEXMakeColormap** to create an X Colormap with the Visual described by *vis_info*. It uses the provided PEX color approximation table entry (in *capx_info*) as a guide in initializing the Colormap contents.

This function modifies only the cells in the region of the Colormap described by the color approximation entry from their default values. The function initializes these cells with read-only permission to improve the chances of sharing colors.

If successful, this function returns the new Colormap ID in *colormap_id_return*. The function does not create or alter any X server properties.

If you specify a color approximation entry of type `PEXColorRange`, the function initializes the colormap to contain a gray ramp.

Errors

See function return values, also:

BadAlloc	The implementation failed to allocate the requested resource.
BadValue	The specified visual is invalid.

See Also

PEXSelectVisual

PEXMatchRenderingTargets - Return Information about Supported Rendering Targets (5.1)

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** or **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 [§2.17.1, page 45].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXMatchRenderingTargets** to get information about which drawable types the implementation supports.

Specify with the *drawable* argument only the screen about which you want information. This function does not apply any of the other drawable attributes.

Specify with the *depth* argument the depth about which you want information. If you want information on all supported depths, then specify a value of zero.

Specify with the *type* argument the type of drawable about which you want information. The type field of the `PEXRenderingTarget` data structure has these same values with the exception of **PEXAnyDrawable**.

Specify with the *visual* argument the visual about which you want information. If you want information on all of the supported visuals, then specify a null pointer.

Indicate with the *max_targets* argument the number of targets that this function should return. If this number is less than the number of matching targets, then it is implementation-dependent which of the matching targets this function returns. If you want this function to return all matching targets, then specify a value of zero.

PEXlib allocates memory for the returned target values. Invoke the **XFree** function to deallocate the memory.

Errors

BadDrawable	The specified drawable resource identifier is invalid.
BadValue	The specified *visual* is invalid or the *type* argument is not one of the defined types.

See Also

PEXQueryColorApproximation - Find a Supported Color Approximation Table Entry for the Specified Example Drawable (5.2)

Synopsis

int **PEXQueryColorApproximation**(Display **display*, Drawable *example_drawable*,
`PEXColorApproxEntry` **color_approx*, unsigned long **count_return*,
`PEXColorApproxEntry` ***color_approx_return*, int **all_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
example_drawable	This example drawable indicates which visual, root, and depth the implementation should use when determining support of the color approximation table entry in question.
color_approx	A pointer to the desired color approximation table entry [§2.16.3, page 39]. The implementation determines if this color approximation table entry is supported for drawables of the same visual, root, and depth as the example drawable.
count_return	Returns the number of returned alternative supported color approximation table entries. A function return value of **True** indicates that this argument does not contain a valid value.
color_approx_return	Returns a list of alternative supported color approximation table entries [§2.16.3, page 39]. A function return value of **True** indicates that this argument does not contain a valid value.
all_return	A returned value of **True** indicates that the list returned in *color_approx_return* is the complete list of supported color approximation table entries. A function return value of **True** indicates that this argument does not contain a valid value.

Returns

True if the given color approximation table entry is supported; **False** otherwise. If an error condition was not identified and **False** is returned, then the output arguments of the function provide alternative color approximation table entries.

Description

Invoke **PEXQueryColorApproximation** to find a supported color approximation table entry for the specified example drawable.

This function returns a value of **True**, if there are drawables that support the specified color approximation table entry, and these drawables have the same visual (if the drawable has a visual), root, and depth as the specified example drawable.

If there are no drawables that support the specified color approximation color table, then this function returns a value of **False**. In this case, this function returns via the *color_approx_return* argument, a list of color approximations which are supported by drawables of the same visual (if the drawable has one), root and depth as the specified example drawable. A return value of **True** on the *all_return* argument indicates that this list is exhaustive.

PEXlib allocates memory if the implementation supports any alternative color approximation table entries. If PEXlib is unable to allocate the memory it requires, then PEXlib sets the *count_return* argument to zero and sets the *color_approx_return* argument to NULL. Invoke the **XFree** function to deallocate this memory.

Errors

BadDrawable	The specified drawable resource identifier is invalid.
BadMatch	The specified drawable is not supported.
BadValue	The type, model or dither members of the `PEXColorApproximation` structure are invalid.

PEXSelectPixmapDepth - Choose a Pixmap Depth for Use with PEXlib Applications (Utility) (5.2)

Synopsis

int **PEXSelectPixmapDepth**(Display **display*, int *screen*, int *criteria_count*, `PEXUtPixmapDepthCriteria` **criteria*, int **depth_return*, `PEXColorApproxEntry` **capx_info_return*, int **select_criteria_return*, unsigned int **unmet_criteria_return*)

Arguments

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

screen — The screen number of the display for which a pixmap depth is to be chosen.

criteria_count — The number of `PEXUtPixmapDepthCriteria` structures in the array pointed to by the *criteria* argument.

criteria — A pointer to a set of `PEXUtPixmapDepthCriteria` structures in priority order from high to low specifying the criteria to be used in selecting the pixmap depth. See the section entitled "Description" below for a complete description of how to use the data structure [§2.33.2, page 82].

depth_return — A pointer to an integer allocated by the client. On a successful return, this function stores the pixmap depth value that best satisfies the criteria in this location.

capx_info_return — A pointer to a `PEXColorApproxEntry` structure allocated by the client. On a successful return, this structure is written with a color approximation setup that is compatible with the pixmap depth and the color approximation type (if specified as a criterion), and can be used in a **PEXSetTableValues** call [§2.16.3, page 39].

select_criteria_return
A pointer to an index into the list of criteria specified indicating the criteria selected, to which the other return arguments correspond. The visual, colormap and color approximation information returned will correspond to the criteria indexed by this return argument. An index value of 0 references the first criteria and index value of criteria_count - 1 references the last criteria.

unmet_criteria_return
A pointer to a mask that describes the criteria that were not satisfied in the selected criteria referenced by *select_criteria_return*. This return argument is meaningful when the function return value is any of *PEXUtSuccess*, *PEXUtQualifiedSuccess*, or *PEXUtCriteriaFailure*.

Returns

PEXUtSuccess — Indicates that the function was completely successful in finding a pixmap depth that meets all hard and soft criteria of the selected depth and encountered no errors during X or PEX inquiries.

PEXUtQualifiedSuccess — Indicates that the chosen depth satisfies all hard criteria but does not meet all soft criteria of the selected criteria. In this case, *unmet_criteria_return* mask includes the soft criteria that were not satisfied.

PEXUtCriteriaFailure — Indicates that no depth could be found that meets all the hard criteria. In this case the *unmet_criteria_return* mask includes the hard and soft criteria that were not satisfied in the first criteria set specified.

PEXUtXFailure — Indicates that an error was encountered during an Xlib function call.

PEXUtPEXFailure — Indicates that an error was encountered during a PEXlib call. One possible cause of this failure is that PEXInitialize has not been called for the display connection.

PEXUtBadAlloc — Indicates that the function ran out of memory.

Description

Invoke **PEXSelectPixmapDepth** as a utility that attempts to select a pixmap depth on the specified display and screen that meets the criteria of one of the criteria sets specified.

This utility returns a success/failure indicator and if successful, it returns information about the pixmap depth and PEX color approximation support for the pixmap depth in structure storage provided by the caller.

PEXSelectPixmapDepth supports a “degradation” sequence of criteria sets: specify the “preferred” characteristics in the first element of the criteria array, and progressively weaker or less desirable sets in subsequent elements. If the hard criterion from the first element of the criteria array is not met, then the utility evaluates the next criteria set.

The function returns color approximation information in *capx_info_return*. When no color approximation type is specified as a criterion, then this utility returns a suitable `PEXColorSpace` entry.

Criteria for pixmap depth selection are specified by two masks inside the criteria structure: [§2.33.2, page 82]

hard_criteria_mask - indicates what fields in the criteria structure contain valid values to be considered absolute (“hard”) criteria during depth selection.

soft_criteria_mask - contains bits for criteria that are desirable but not absolutely required for success.

If a bit for a particular criterion is present in both masks then it is treated as a hard criterion.

For each bit that is set in a criteria mask, a corresponding field in the `PEXUtPixmapDepthCriteria` structure is expected to be set to appropriate values:

PEXUtDepth	`depth`
PEXUtColorApproxType	`color_approx_type`
PEXUtZBufferDepth	`z_buffer_depth`
PEXUtAlphaBufferDepth	`alpha_buffer_depth`
PEXUtAccumBufferDepth	`accum_buffer_depth`
PEXUtAccumBufferMinRed	`accum_buffer_min_red`
PEXUtAccumBufferMinGreen	`accum_buffer_min_green`
PEXUtAccumBufferMinBlue	`accum_buffer_min_blue`

Depth refers to the depth of the pixmap.

Color_approx_type is mainly useful to applications that need PEXColorRange color approximation. If not specified, then this utility assumes that PEXColorSpace is acceptable, since this is the most common and natural method for most PEX applications. If it is set to PEXColorRange, the returned color approximation entry is initialized for an appropriate gray ramp.

Use the Z buffer depth criterion to specify a minimum depth criterion in terms of bits. If you intend to read the contents of a Z buffer, the function “PEXGetAncillaryBufferInfo - Get Ancillary Buffer Information (5.2)” on page 97 is useful for determining the format and range of the Z buffer values.

Specify alpha buffer depth as a criterion if you expect to use alpha blending functions that utilize a destination alpha buffer or intend to use alpha buffer transparency.

Specify some combination of the four accumulation buffer criteria if you intend to use accumulation buffers for multi-pass rendering.

The ability of this procedure to verify that criteria are met depends on information available from the PEX server, including its support for various interoperability conventions. In general, if a hard criterion cannot be determined to have been satisfied or not, it is considered to have not been met; if a soft criterion cannot be verified, then the utility acts as though it had been met but still returns the corresponding bit in *unmet_criteria_return*.

If two or more pixmap depths meet all the hard criteria, then the depth that meets the largest number of soft criteria is selected. If there is more than one depth that meets the same number of soft criteria, then the depth with the highest color resolution is chosen. The final criterion is that preference is given to visuals in the order that they are listed by the server.

Errors

See function return values.

See Also

PEXMakeColormap, PEXMatchRenderingTargets, PEXSelectVisual

PEXSelectVisual - Choose a Visual for Use with PEXlib Applications (Utility) (5.2)

Synopsis

int **PEXSelectVisual**(Display **display*, int *screen*, int *criteria_count*, `PEXUtVisualCriteria` **criteria*, `XVisualInfo` **vis_info_return*, `XStandardColormap` **cmap_info_return*, `PEXColorApproxEntry` **capx_info_return*, int **select_criteria_return*, unsigned int **unmet_criteria_return*, Atom **std_prop_atom_return*)

Arguments

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

screen The screen number of the display for which a Visual is to be chosen.

criteria_count The number of `PEXUtVisualCriteria` structures in the array pointed to by the *criteria* argument.

criteria A pointer to a set of `PEXUtVisualCriteria` structures in priority order from high to low specifying the criteria to be used in selecting the Visual.
See the section entitled "Description" below for a complete description of how to use the data structure [§2.33.1, page 82].

vis_info_return A pointer to an `XVisualInfo` structure allocated by the client. On successful return, this structure is written with a description of the chosen Visual.

cmap_info_return A pointer to an `XStandardColormap` structure allocated by the client. On successful return, this structure is written with a copy of the standard colormap property entry that was used to choose the Visual, if `PEXUtStandardColormapProperty` was specified in the hard or soft criteria mask and such an entry was found. If no property was found or the criteria masks do not include `PEXUtStandardColormapProperty`, this structure is written with information derived from the attributes of the Visual.

capx_info_return A pointer to a `PEXColorApproxEntry` structure allocated by the client. On a successful return, this structure is written with a color approximation setup that is compatible with the *cmap_info_return* contents, the Visual, and the color approximation type (if specified as a criterion), and can be used in a **PEXSetTableValues** call [§2.16.3, page 39].

select_criteria_return
A pointer to an index into the list of criteria specified indicating the criteria selected, to which the other return arguments correspond. The visual, colormap and color approximation information returned will correspond to the criteria indexed by this return argument. An index value of 0 references the first criteria and index value of criteria_count - 1 references the last criteria.

unmet_criteria_return
A pointer to a mask that describes the criteria that were not satisfied in the selected criteria referenced by *select_criteria_return*. This return argument is meaningful when the function return value is any of *PEXUtSuccess*, *PEXUtQualifiedSuccess*, or *PEXUtCriteriaFailure*.

std_prop_atom_return
A pointer to an Atom. Upon successful return, this location contains the Atom value for the property whose entry is described in **cmap_info_return*, or **None** if no property entry was used.

Returns

PEXUtSuccess Indicates that the function was completely successful in finding a Visual that meets all hard and soft criteria of the selected Visual and

	encountered no errors during X or PEX inquiries.
PEXUtQualifiedSuccess	Indicates that the chosen Visual satisfies all hard criteria but does not meet all soft criteria of the selected criteria. In this case, *unmet_criteria_return* mask includes the soft criteria that were not satisfied.
PEXUtCriteriaFailure	Indicates that no Visual could be found that meets all the hard criteria. In this case the *unmet_criteria_return* mask includes the hard and soft criteria that were not satisfied in the first criteria set specified.
PEXUtXFailure	Indicates that an error was encountered during an Xlib function call.
PEXUtPEXFailure	Indicates that an error was encountered during a PEXlib call. One possible cause of this failure is that PEXInitialize has not been called for the display connection.
PEXUtBadAlloc	Indicates that the function ran out of memory.

Description

Invoke **PEXSelectVisual** as a utility that attempts to select an X Visual on the specified display and screen that meets the criteria of one of the criteria sets specified.

This utility returns a success/failure indicator and if successful, it returns information about the Visual and PEX color support for the Visual in structure storage provided by the caller.

PEXSelectVisual supports a "degradation" sequence of criteria sets: specify the "preferred" Visual in the first element of the criteria array, and progressively weaker or less desirable sets in subsequent elements. If the hard criterion from the first element of the criteria array is not met, then the utility evaluates the next criteria set.

The function returns colormap initialization information and color approximation information in *cmap_info_return* and *capx_info_return*. When no color approximation type is specified as a criterion, and no property is available to describe a colormap, then this utility returns a `PEXColorRange` color approximation entry for GrayScale and StaticGray Visuals, and a `PEXColorSpace` entry otherwise.

If a standard colormap property was used during Visual selection, then the Atom is returned in *std_prop_atom_return*. All storage for returned values must be allocated by the caller.

Criteria for Visual selection are specified by two masks inside the criteria structure: [§2.33.1, page 82]

hard_criteria_mask - indicates what fields in the criteria structure contain valid values to be considered absolute ("hard") criteria during Visual selection.

soft_criteria_mask - contains bits for criteria that are desirable but not absolutely required for success.

If a bit for a particular criterion is present in both masks then it is treated as a hard criterion.

For each bit that is set in a criteria mask, a corresponding field in the `PEXUtVisualCriteria` structure is expected to be set to appropriate values:

PEXUtDepth	`depth`
PEXUtMinColors	`min_colors`
PEXUtMinRed	`min_red`
PEXUtMinGreen	`min_green`
PEXUtMinBlue	`min_blue`
PEXUtVisualClass	`visual_class`
PEXUtStandardColormapProperty	`standard_colormap_property`
PEXUtDynamicVisual	`dynamic_visual`
PEXUtDBCapability	`double_buffering_capability`
PEXUtColorApproxType	`color_approx_type`

PEXUtZBufferDepth	`z_buffer_depth`
PEXUtAlphaBufferDepth	`alpha_buffer_depth`
PEXUtAccumBufferDepth	`accum_buffer_depth`
PEXUtAccumBufferMinRed	`accum_buffer_min_red`
PEXUtAccumBufferMinGreen	`accum_buffer_min_green`
PEXUtAccumBufferMinBlue	`accum_buffer_min_blue`

The depth, min_colors, and (min_red,min_green,min_blue) criteria fields interact with one another, and typically only some subset of these should be specified. Depth refers to the depth of the Visual, and may be combined with visual_class if the application must have a specific Visual type (e.g., depth 8 PseudoColor). The depth field may have any valid depth (number of planes) as a value; the value of the visual_class field must be one of the standard X Visual classes (PseudoColor, DirectColor, GrayScale, StaticColor, TrueColor, StaticGray). Min_colors specifies a minimum number of total colors (i.e., cells in a PseudoColor Visual, or combinations in a DirectColor Visual). Min_red, min_green, and min_blue can be used to indicate more specific requirements of a DirectColor or TrueColor visual, or of a color space sampling specified in a standard colormap property. Allowed values for all these fields are positive integers.

When the standard_colormap_property criterion is specified (allowed values are **True** and **False**), the procedure searches for defined standard colormap properties in a predefined order: first RGB_BEST_MAP, then RGB_DEFAULT_MAP. It is recommended that standard_colormap_property be specified as a "soft" criterion, since some PEX servers do not define standard colormap properties.

This utility may choose a Visual with more than the requested number of colors unless some other criterion prevents it from doing so.

You can use the dynamic_visual criterion as a hard criterion to enforce selection of a Visual that you can write to, or as a soft criterion to indicate a preference. Conflict is possible between this criterion and the visual_class field. Allowed values are **True** and **False**.

The double_buffering_capability criterion may be important to many interactive graphics applications. Allowed values are **True** and **False**.

Color_approx_type is mainly useful to applications that need PEXColorRange color approximation. If not specified, then this utility assumes that PEXColorSpace is acceptable, since this is the most common and natural method for most PEX applications. If it is set to PEXColorRange, the returned color approximation entry is initialized for an appropriate gray ramp.

Use the Z buffer depth criterion to specify a minimum depth criterion in terms of bits. If you intend to read the contents of a Z buffer, the function "PEXGetAncillaryBufferInfo - Get Ancillary Buffer Information (5.2)" on page 97 is useful for determining the format and range of the Z buffer values.

Specify alpha buffer depth as a criterion if you expect to use alpha blending functions that utilize a destination alpha buffer or intend to use alpha buffer transparency.

Specify some combination of the four accumulation buffer criteria if you intend to use accumulation buffers for multi-pass rendering.

For good application portability and interoperability, the following recommendation is made: always specify standard_colormap_property as a soft criterion. If the application has a need for a certain amount of color resolution, specify either min_colors or (min_red,min_green,min_blue) but not both. Double_buffering_capability may be specified as hard or soft criteria depending on the application. Other criteria should be specified as soft criteria unless particular application needs cause them to be absolute requirements.

The ability of this procedure to verify that criteria are met depends on information available from the PEX server, including its support for various interoperability conventions. In general, if a hard criterion cannot be determined to have been satisfied or not, it is considered to have not been met; if a soft criterion cannot be verified, then the utility acts as though it had been met but still returns the corresponding bit in *unmet_criteria_return*.

If two or more visuals meet all the hard criteria, then the visual that meets the largest number of soft criteria is selected. If there is more than one visual that meets the same number of soft criteria, then the visual with the highest color resolution is chosen. In the event that the visuals are equivalent in these three respects, a static visual is preferred over a dynamic one. The final criterion is that preference is given to visuals in the order that they are listed by the server.

Errors

See function return values.

See Also

PEXMakeColormap, PEXMatchRenderingTargets, PEXSelectPixmapDepth

PEXSetAltProtocolLevel - Set Alternate Protocol Level Generation

Synopsis

void **PEXSetAltProtocolLevel**(Display **display*, int *level*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
level	The alternate protocol level (**PEXProtocolLevel5_1**, **PEXProtocolLevel5_2**)

Returns

None.

Description

Invoke **PEXSetAltProtocolLevel** to set the alternate protocol level that PEXlib uses to generate protocol when it is possible to encode a request or output command with an older variety of the protocol element.

The default alternate protocol level is **PEXProtocolLevel5_2**.

Use this function to set a lower alternate protocol level when you are designing a PEXlib application that is intended to work with that lower protocol level. You must, of course, restrict the application to use only the features at that protocol level. However, some PEXlib functions are capable of generating protocol for the intended protocol element at different protocol levels and do so at the highest level, unless instructed differently by this function.

For example, there are two different protocol encodings for the TriangleStrip output command. Normally, the **PEXOCCTriangleStrip** function encodes the primitive with the 5.2 version of the OC. If, for some reason, you need to force PEXlib to generate the 5.1 encoding, use this function to set the alternate protocol level to **PEXProtocolLevel5_1**. Take care to specify only the TriangleStrip data that is supported by PEX 5.1, otherwise an error may occur.

You must invoke **PEXInitialize** prior to using this function. If you specify an invalid value for *level*, the function ignores that value and the previous alternate protocol level remains in effect.

Errors

None.

See Also

Chapter 4

Programming Interface for Generating Output Commands

4.1. Rendering and Storing Output Commands

The functions described in this chapter generate output commands. You can direct these output commands to a renderer for immediate processing or you can store them in a structure resource.

PEXlib does not directly expose the specific protocol requests **PEXRCStoreElements** and **PEXRCRenderOutputCommands**. Instead, the request type argument, *req_type*, determines which of these two requests to use and when to use them to send the output commands to the PEX implementation. This argument must have one of four values: **PEXOCRender**, **PEXOCRenderSingle**, **PEXOCStore** or **PEXOCStoreSingle**. **PEXOCRender** and **PEXOCRenderSingle** cause PEXlib to format the output command into a **PEXRCRenderOutputCommands** request. This request causes the implementation to immediately render the output commands using the specified renderer when the implementation receives the request. **PEXOCStore** and **PEXOCStoreSingle** cause PEXlib to format the output command into a **PEXRCStoreElements** request. This request causes the implementation to store the output commands into the specified structure when the implementation receives the request. If the request type argument is not one of the values defined above, then the result is implementation-dependent.

To improve performance, PEXlib tries to pack output commands into single **PEXRCRenderOutputCommands** or **PEXRCStoreElements** requests. This packing, or buffering, of output commands normally occurs when the request type is **PEXOCRender** or **PEXOCStore**. When the request type is **PEXOCRenderSingle** or **PEXOCStoreSingle**, PEXlib forces the output command to be formatted into a single request by itself. This is useful, for example, for debugging purposes.

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 14* for more detail. The result of the **PEXRCStoreElements** request when the specified structure's editing mode is set to **PEXStructureReplace** is unpredictable when using the request type **PEXOCStore**. The **PEXRCStoreElements** request replaces a range of elements in the structure, depending on how many output commands are in the request.

The number of output commands in a given request depends on too many variables to predict. PEXlib will not pack a single request that is too large to send to the server. However, if a single OC is too large to send, PEXlib causes the PEX server to issue a *BadLength* error. Invoke the **PEXGetSizeOCs** and **XMaxRequestSize** functions beforehand to determine if an OC is too large to send. See *Chapter 1:Introduction to PEXlib* for more detail.

There are also a number of OC manipulation functions defined in "Output Command Encode and Decode" on page 725 and in "Sending Encoded Output Commands" on page 731.

For a PHIGS workstation resource to process output commands for rendering, you must store the output commands in structures. The PHIGS workstation subset does not, in itself, support immediate mode rendering. See *Chapter 19:PHIGS Workstations* for more information on PHIGS workstations.

4.2. PEXlib Explicit Interface

PEXlib offers two major argument interfaces for the output command functions: the *explicit interface* and the *output command context*, or OC Context (OCC) interface.

The *explicit interface* is the interface defined for PEXlib 5.1 and is included in the PEXlib 5.2 specification for backwards compatibility, so PEXlib 5.1 programs compile with PEXlib 5.2 libraries. Although you may still use the explicit interface, you are encouraged to use the OCC interface to take advantage of the performance and functional improvements available in PEXlib 5.2. The explicit interface uses the same first three arguments:

display - specifies the display connection.

resource_id - specifies the resource identifier for the targeted renderer (for displaying output commands immediately) or structure (for storing output commands).

req_type - specifies whether the application renders the output commands immediately (in which case the *resource_id* argument identifies the renderer resource), or stores the output commands in a structure (in which case the *resource_id* argument identifies the structure resource).

The explicit interface requires that you specify all arguments on every output command function you invoke.

4.3. PEXlib Output Command Context (OCC) Interface

The *output command context* interface (introduced with PEXlib 5.2) requires that the first argument always be the OC context, replacing the three arguments listed above. The OC context is an opaque structure that contains many of the arguments that are commonly found in the output command functions. Your application uses a set of special OCC manipulation functions to modify the fields in the opaque OC context. Then your application uses the OC Context in subsequent invocations of output command functions. The OCC interface uses far fewer arguments than the explicit interface, making coding easier and improving performance in some implementations. See "Output Command Context (OC Context or OCC)" on page 134 for the complete description of the OCC.

Note: the attribute and primitive output commands introduced after 5.1 do not offer the explicit argument interface format. This is to encourage the use of the OCC argument interface in newer applications over the older explicit interface.

4.3.1. Flexible Data Formats

The OCC style functions for primitives that have facet and/or vertex data parameters allow you to supply the graphical data (coordinates, vertex attributes, facet attributes, and floating-point data) in *packed, stride, or unpacked* form. The *packed* form is the only form supported in PEXlib 5.1 and requires you to format the data into packed data structures defined by PEXlib. The *stride* form is more flexible and allows you to supply data formatted in application-defined structured arrays[1] without the need to copy the data into the PEXlib-defined structures before invoking the PEXlib function. See "Facet / Vertex Data Formats" on page 137. The *unpacked* form allows you to supply the data in separate lists for each data type. See "Unpacked Data Interface" on page 141.

4.3.2. Data Alignment

On many machine architectures, data alignment can be very important for performance and correct execution. For example, some machines require that pointers point to word boundaries when accessing word-length data items. Because the OCC interface allows you to control the offset applied to pointers for data accesses, you should be extra careful when specifying OCC pointers and offsets to ensure that you are following your machine's data alignment policies.

4.4. Output Command Context (OC Context or OCC)

The OC Context maintains the state of some common PEXlib arguments across PEXlib function calls. The OC context enables you to set the desired values of these arguments and then reuse them by specifying just the OC Context in several subsequent function calls. This eliminates the need to re-specify these same arguments in every function call, reducing redundancy and improving performance in some environments.

The OC Context itself is an opaque data structure with members that are referred to as values. Programs cannot alter the

1. A "structured array", in this context, refers to an array of C language structures (as in struct) which is a common idiom employed in C language data structures, for example:

```
struct MyStruct { float x,y; } myData[30];
```

values of the OC Context directly. Instead, programs must use additional function calls to manipulate the values of the context. This ensures that the PEXlib implementation is informed whenever an OC Context value is changed.

4.4.1. Data Structures

The primary data structure used with the OC Context is the `PEXOCCValues` data structure which you use to set the value fields via the bitmask/value mechanism. It is important to realize that you do not alter the OC Context itself. You instead specify your desired changes in the `PEXOCCValues` data structure and use OCC functions to alter the OC Context with this data structure as input. The default values assigned to the OC Context, when your application creates it, are also given below:

```
typedef struct {                                    /* DEFAULT Values */
      Display            *display;                  /* undefined */
      PEXRenderer        renderer;                  /* undefined */
      PEXStructure       structure;                 /* undefined */
      PEXOCRequestType   req_type;                  /* PEXOCRender */
      int                shape_hint;                /* PEXShapeUnknown */
      int                ignore_edges;              /* False */
      int                contour_hint;              /* PEXContourUnknown */
      int                contours_all_one;          /* False */
      unsigned int       facet_attributes;                /* PEXGANone */
      unsigned int       line_vertex_attributes;          /* PEXGANone */
      unsigned int       marker_vertex_attributes;        /* PEXGANone */
      unsigned int       surface_vertex_attributes;       /* PEXGANone */
      unsigned int       edge_attributes;                 /* PEXGANone */
      unsigned int       facet_fp_data_count;             /* 0 */
      unsigned int       line_vertex_fp_data_count;       /* 0 */
      unsigned int       marker_vertex_fp_data_count;     /* 0 */
      unsigned int       surface_vertex_fp_data_count; /* 0 */
      int                color_type;                /* PEXColorTypeRGB */
      char               *encoding_state;           /* NULL Pointer */
      int                data_model;                /* PEXDataPacked */
      union {
            PEXOCCStrideData         stride;
            PEXOCCUnpackedData       unpacked;
      } data_model_specs;
} PEXOCCValues;

typedef struct {                                    /* all members 0 */
      int                facet_stride;
      int                vertex_stride;
      int                facet_color_offset;
      int                facet_normal_offset;
      int                facet_fp_data_offset;
      int                vertex_coord_offset;
      int                vertex_color_offset;
      int                vertex_normal_offset;
      int                vertex_edge_offset;
      int                vertex_radius_offset;
      int                vertex_axes_offset;
      int                vertex_angle_offset;
      int                line_vertex_fp_data_offset;
      int                marker_vertex_fp_data_offset;
      int                surface_vertex_fp_data_offset;
      } PEXOCCStrideData;
```

```
typedef struct {                                   /* all members 0 */
      int               facet_color_size;
      int               facet_normal_size;
      int               facet_fp_data_count;
      int               vertex_coord_size;
      int               vertex_color_size;
      int               vertex_normal_size;
      int               vertex_edge_size;
      int               vertex_radius_size;
      int               vertex_axes_size;
      int               vertex_angle_size;
      int               line_vertex_fp_data_count;
      int               marker_vertex_fp_data_count;
      int               surface_vertex_fp_data_count;
      } PEXOCCUnpackedData;
```

The semantics of each member in the PEXOCCValues structure is the same as it is in the explicit interface.

Specify all size and offset members in terms of bytes. The `fp_data_count` members represent the *number* of floating point values.

The *encoding_state* member is reserved for use by future internationalized text functions to retain character encoding state for languages that require it.

The *data_model* member indicates the data model used to pass vertex and facet data to OCC primitive functions. You should set this member to **PEXDataPacked**, **PEXDataStride**, or **PEXDataUnpacked**.

The OCC itself is defined as:

```
typedef struct _PEXOCC
#ifdef PEXLIB_ILLEGAL_ACCESS
{
    XExtData     *ext_data;          /* hook for extension  */
    /* PEXlib private data */
}
#endif
*PEXOCC;
```

You cannot access any members within the OCC directly. You must simply pass the OCC to functions that modify or use the contents of the context structure.

4.4.2. Sample Usage of the OCC

To invoke the OC functions that use the OC Context, your application must first create an OC Context with the **PEXCreateOCC** function. This function returns an OC Context initialized with default values for future reference. The application can change the default context values when it creates the OC Context and/or can change values later by invoking the **PEXChangeOCC** function. These two functions set the value fields via a bitmask/value mechanism and are suitable when you are setting several values at once. There are also OC Context convenience functions that, through a simpler interface, enable you to set only one value per function in the OCC.

Here is a coded example of how to use the OC Context. The last two statements illustrate the difference between an OCC function and a non-OCC function:

```
PEXOCCValues        ocvalues;
unsigned int        mask;
PEXOCC              myocc;

ocvalues.display = my_display;
PEXSetOCCValueMask(&mask, PEXOCCMDisplay);
ocvalues.renderer = my_renderer;
PEXSetOCCValueMask(&mask, PEXOCCMRenderer);
ocvalues.color_type = PEXColorTypeRGB;
PEXSetOCCValueMask(&mask, PEXOCCMColorType);
ocvalues.surface_vertex_attributes = PEXGAColor;
PEXSetOCCValueMask(&mask, PEXOCCMSurfaceVertexAttributes);

myocc = PEXCreateOCC(&mask, &ocvalues);

PEXOCCTriangleStrip(myocc, NULL, count, vertices);

/* This is the old, explicit interface */
PEXTriangleStrip(display, resource_id, req_type, PEXGANone,
 PEXGAColor, 0, 0, PEXRGBFloat, NULL, count, vertices);
```

Note that it was not necessary to set the facet attributes member of the OCC to **PEXGANone** because **PEXGANone** is the default value in a newly created OCC.

Invoke the **PEXFreeOCC** function to deallocate the memory associated with the OC Context.

4.5. Facet / Vertex Data Formats

There are three ways to provide facet and vertex data to PEXlib:

4.5.1. Packed Data Interface (PEXlib 5.1 method)

Unlike the stride and unpacked interfaces, you can use the packed data interface with either OCC primitive functions or with non-OCC primitive functions. The use of the packed data interface with non-OCC functions, as with PEXlib 5.1 functions, is as follows (this example has no facet data and supplies colors and normals in the vertex data):

```
PEXVertexRGBNormal      my_vertices[20];
PEXArrayOfVertex        verts;

/* code to init my_vertices */

verts.rgb_normal = my_vertices;

PEXTriangleStrip(display, rend, PEXOCRender, 0,
                 PEXGAColor | PEXGANormal, PEXColorTypeRGB,
                 NULL, 20, verts);
```

You can also use packed data format with OCC style functions, which you may find useful when converting older code to use the OCC style functions:

```
PEXVertexRGBNormal      my_vertices[20];

/* code to init my_vertices */

PEXOCCTriangleStrip(context, NULL, 20, my_vertices);
```

This usage assumes that you have already initialized the OC Context fields with the correct values. In particular, you need to set the *display*, *renderer (or structure)*, *req_type*, *surface_vertex_attributes*, *color_type*, and *data_model* (**PEXDataPacked**, of course) fields. Unless these fields change, you only need to set these once. You do not need to set any fields in the *data_model_specs* union.

Also, you do not need the `PEXArrayOfVertex` union to pass the vertex data in the function. The type of the facet and vertex data parameters is `PEXPointer`, which allows you to pass any type in this parameter. This is important for the stride interface described in the next section. However, for both facet and vertex data, you are responsible for making sure that the data you are passing accurately reflects the attributes that you set in the facet or vertex attribute fields.

PEXlib does not supply a set of structure type definitions for facet or vertex data that include floating point data because there is no way to determine how many floating point numbers you want to supply with a facet or vertex. Therefore, you may find it convenient to design your own data structure to pass in the facet or vertex parameters of some OC functions when using the **PEXDataPacked** data model. You need to design the data structure with the following ordering rules in mind:

For facet data, the required order is:

```
PEXColor*          one of the PEXlib color types, if provided
PEXVector          normal, if provided
float[n]           floating point data, if provided
```

For vertex data, the required order is:

```
PEXCoord           (or PEXCoord2D) center
PEXColor*          one of the PEXlib color types, if provided
PEXVector          normal, if provided
unsigned int       edges, if provided
float[n]           floating point data, if provided
```

As an example, if you wish to supply vertex data for FillArea using 3D coordinates, RGB floating-point colors and three floating-point numbers (for texture data), you would create a structure like this:

```
typedef struct {
      PEXCoord          center;
      PEXColorRGB       rgb;
      float             texture_data[3];
} MY_PEXRGBTextureVertexData;
```

Remember to set the correct surface vertex attributes in the OCC. In this case you would make sure that the **PEXGA2D** flag is off, the **PEXGAColor** is on and the **PEXGAFloatData** flag is on. You set the *color_type* field in the OCC to **PEXColorTypeRGB** and the number of floats in the OCC to three.

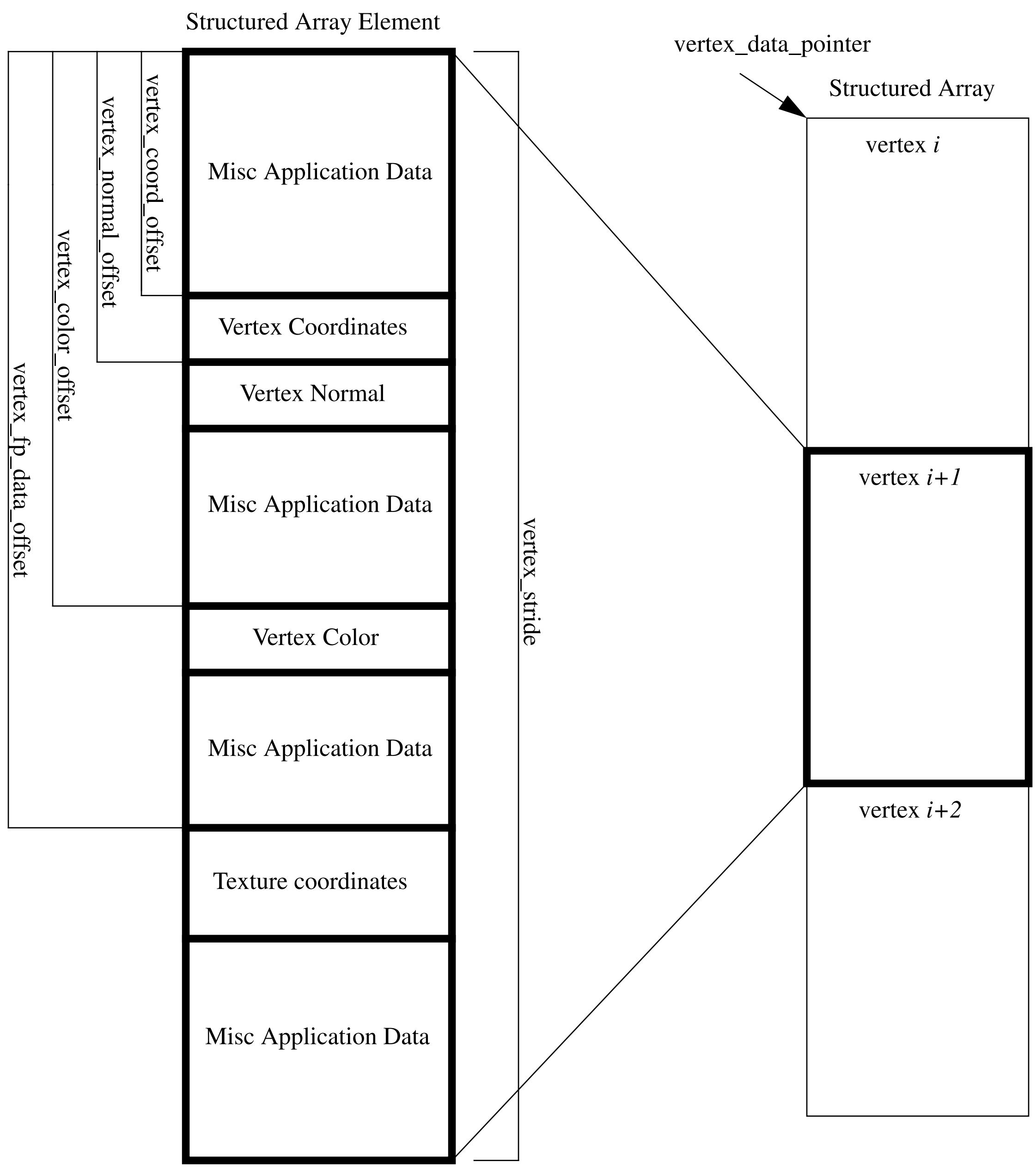
You may then pass the address of an array of these structures directly in the OCC form of a function:

```
MY_PEXRGBTextureVertexData    my_vertices[20];

/* code to init my_vertices */

PEXOCCTriangleStrip(context, NULL, 20, my_vertices);
```

4.5.2. Stride Data Interface

The stride interface allows you to access the facet and vertex data directly from application data structures if the data is arranged in a "structured array" format, where the application stores facet or vertex data in an array of structures; each structure in the structured array corresponds to data for a single facet or vertex along with other application specific data. You specify the size, or "stride" of each array element and the offsets of each of the facet or vertex data items within the array element. Using this format, you eliminate the need to copy the facet or vertex data from the application data structure to an array of PEXlib packed data structures.

Here is an illustration of the stride model, using vertex data:



In this example, you supply a pointer to the beginning of the structured array in the function call.

```
PEXOCCTriangleStrip(context, NULL, 20, vertex_data_pointer);
```

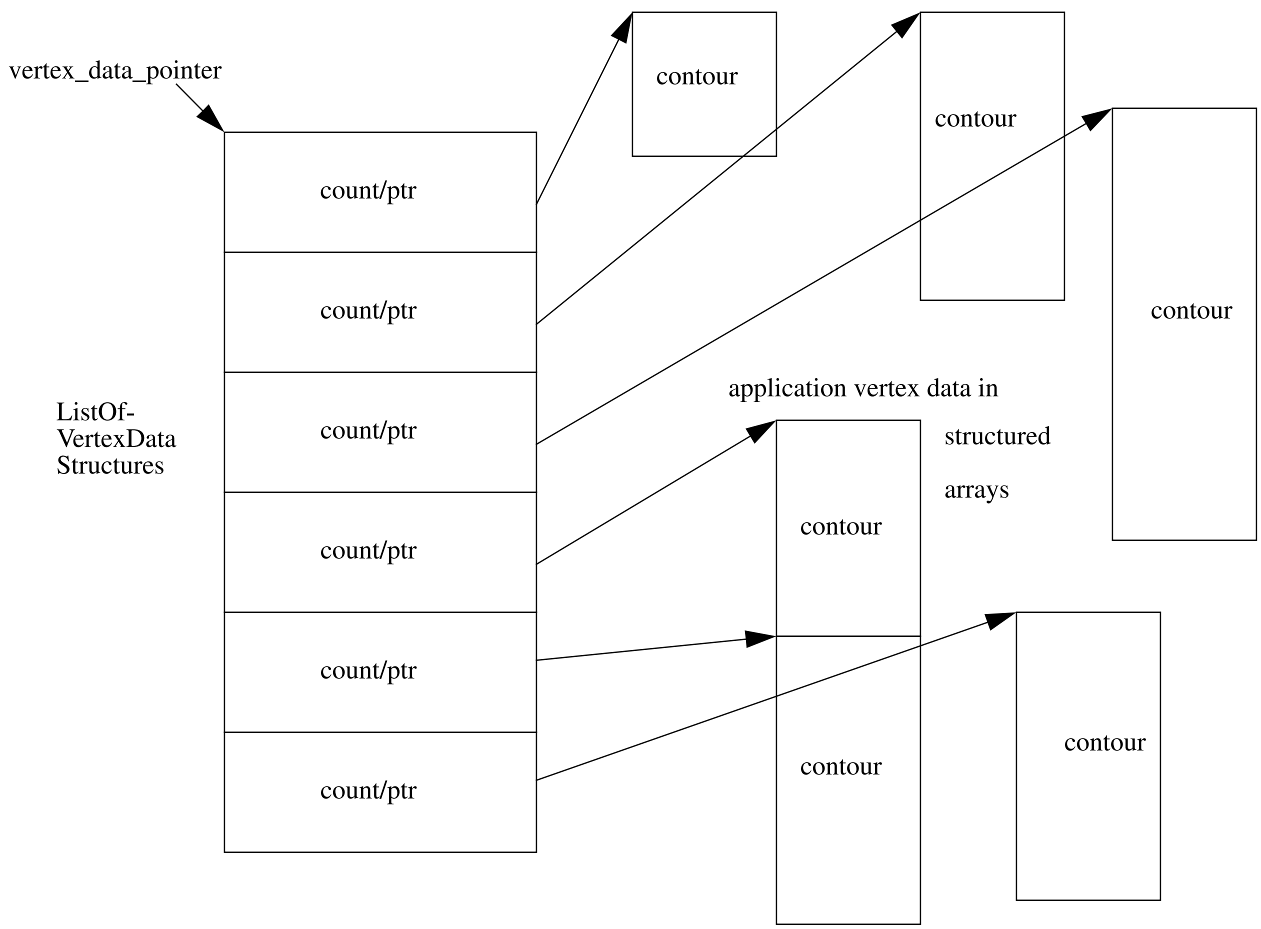
In this case, you set the OCC's *data_model* to **PEXDataStride**, and in addition to the OC Context values you had to set for the packed case, you also need to set *vertex_stride* and the offset members in the *stride* structure used above (in this case, *vertex_coord_offset*, *vertex_normal_offset*, *vertex_color_offset*, and *vertex_fp_data_offset*.) You do not need to set any offset members in the stride structure for data that you don't use. You tell PEXlib what data items it is to use by the facet/vertex attribute OCC members. With the stride interface, you must supply the offset for every data item that you specify is present in the facet/vertex attributes.

You handle facet data in exactly the same way, except you use different fields in the OC Context.

Some primitives, such as Fill Area Set and Polylines (Polyline Set) are defined by a nested list of vertices. Since you cannot work directly with a nested list of vertices, you use the "PEXListOfVertexData Data Structure" on page 50, which looks like:

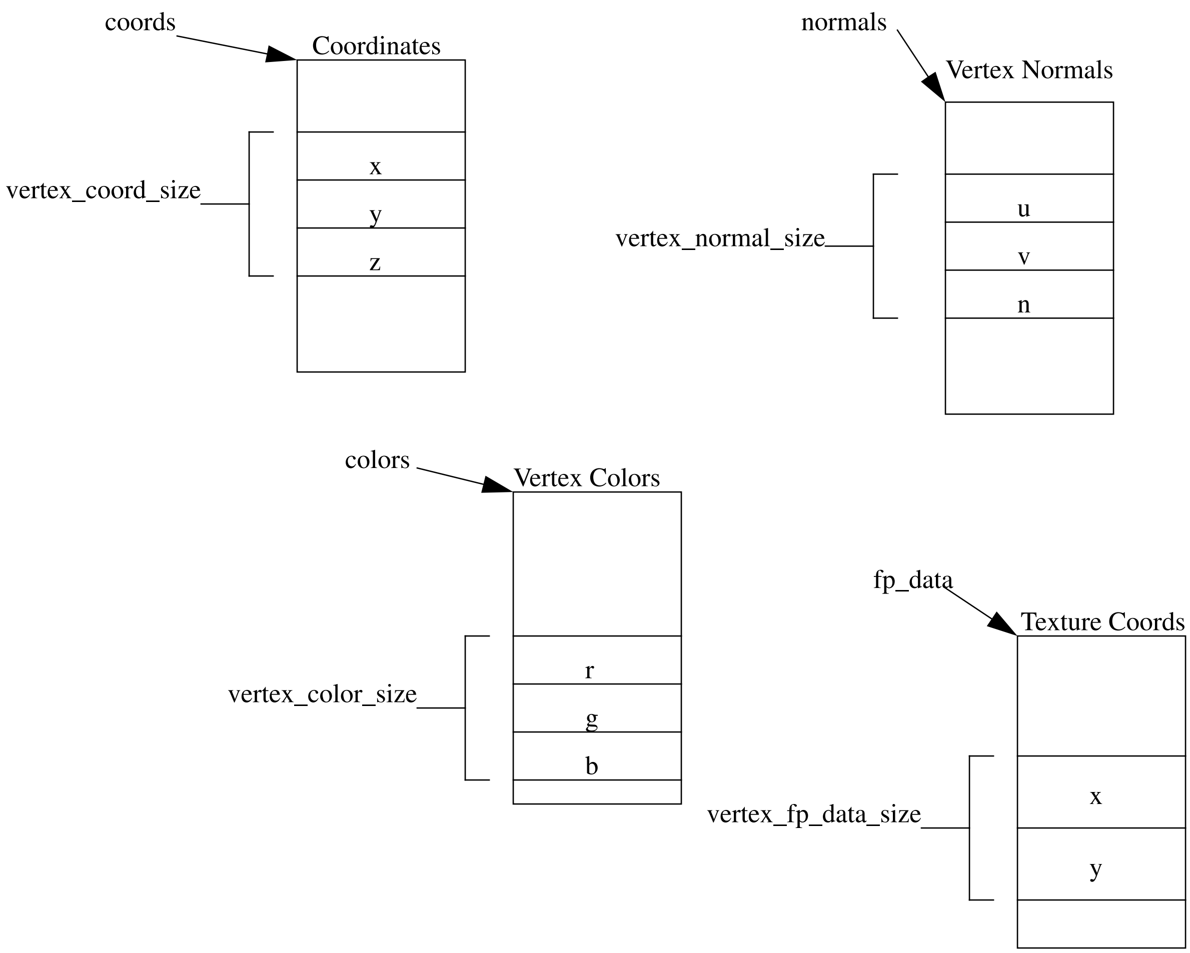
```
typedef struct {
      unsigned long     count;
      PEXPointer        vertices;
} PEXListOfVertexData;
```

For each Fill Area Set Contour or Polyline in the set, you allocate one of these structures and fill it in with the number of points in that contour or line and a pointer to the first vertex in the list. Note that the type of the *vertices* pointer is `PEXPointer`, so you set this pointer to point to a structured array, just as in the non-nested case. The list of these `PEXListOfVertexData` structures must be contiguous and you pass the address of the first one in the vertex data argument of the primitive function call.



4.5.3. Unpacked Data Interface

You use this data interface style when your data is arranged in lists or arrays of points, colors, normals, etc. Each of these lists can reside anywhere in memory, since you supply a pointer to the start of each list. You also must specify the size of each element in the list by using the *unpacked* member of the OC Context.



When using this data format, as in this example, you need to set the *data_model* OCC member to **PEXDataUnpacked** and to set the *vertex_coord_size*, *vertex_normal_size*, *vertex_color_size*, and *vertex_fp_data_count* fields of the OCC. In addition, you pass pointers to each of the data arrays, four of them in this case, in the vertex data parameter of the primitive function call. To pass the four pointers, place them in the following structure and pass the address of the structure:

```
typedef struct {
      PEXPointer  coords;
      PEXPointer  colors;
      PEXPointer  normals;
      PEXPointer  edges;
      PEXPointer  radii;
      PEXPointer  axes;
      PEXPointer  angles;
      PEXPointer  fp_data;
} PEXUnpackedVertexData;
```

An example of using the unpacked data pointers:

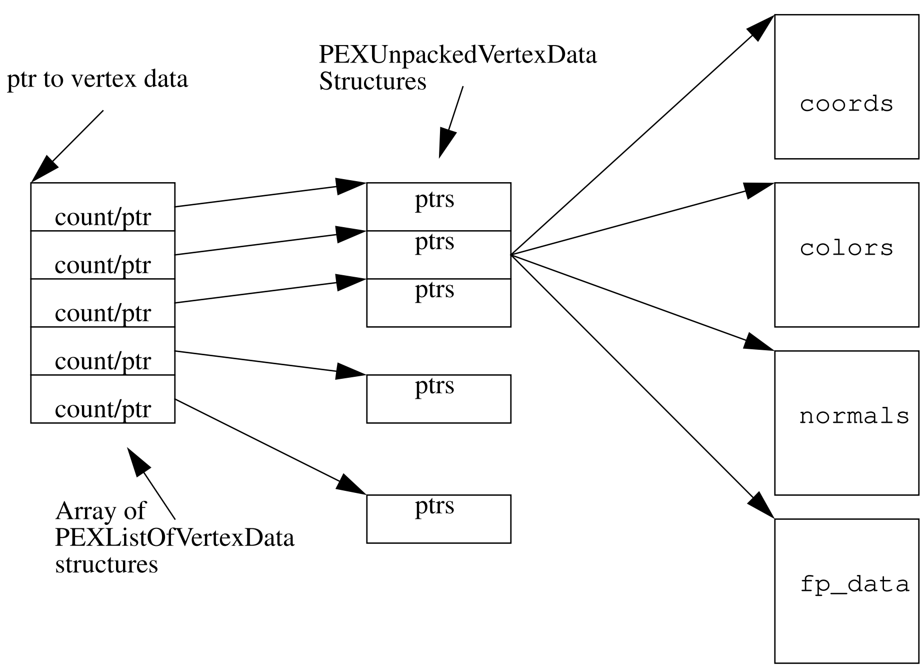
```
PEXUnpackedVertexData         verts;

verts.coords = my_coords;
verts.colors = my_colors;
verts.normals = my_normals;
verts.fp_data = my_fp_data;

PEXOCCTriangleStrip(context, NULL, count, &verts);
```

When working with facet data, use the "PEXUnpackedFacetData Data Structure" on page 51

Some primitives, such as Fill Area Set and Polylines (Polyline Set) are defined by a nested list of vertices. Since you cannot work directly with a nested list of vertices, you use the "PEXListOfVertexData Data Structure" on page 50, which looks like:

```
typedef struct {
      unsigned long     count;
      PEXPointer        vertices;
} PEXListOfVertexData;
```

For each Fill Area Set Contour or Polyline in the set, you allocate one of these structures and fill it in with the number of points in that contour or line and a pointer to the first vertex in the list. Note that the type of the *vertices* pointer is `PEXPointer`, so you set this pointer to point to a `PEXUnpackedVertex` data structure, just as in the non-nested case. The list of these `PEXListOfVertexData` structures must be contiguous and you pass the address of the first one in the vertex data argument of the primitive function call.



4.6. Errors and Output Command Errors

The PEX output commands are designed so that the PEX implementation can often continue to process an OC with an error in it by using defaults or fall-back values. However, in some cases, it is not practical to define a reasonable fall-back value and so the PEX implementation stops processing OC's and generates an error, usually a *BadPEXOutputCommand* error.

Many *BadPEXOutputCommand* errors are very specific to the output command and are listed with the function. There are another set of errors that are common to a number of output commands and are not generally listed with each OC function.

BadPEXOutputCommand: Most cases are listed with each OC. Some of the causes common to many OC's are:

- An argument expected to contain a value from a fixed enumeration contains an undefined value. An example of this is the text_path text attribute. If you stick with the listed possibilities, or possibilities allowed by an extension, then you should not get errors.
- The color data is not in a format supported by the PEX implementation. Since a large number of output commands use color data, this error is not listed with each OC function.
- Setting a bit in a bitmask to one when that bit is defined as unused. To avoid errors, set only bits that have been defined by PEXlib, though some PEX 5.1 implementations may tolerate the setting of undefined bits.

BadPEXStructurePermission: If you attempt to store an output command in a locked structure, the implementation generates this error.

Chapter 5

OC Context Functions

5.1. Bitmask/Value Interface

Several of the functions described in this chapter utilize a bitmask/value approach to specifying the value of individual fields in the OC Context. The approach is very similar to the one used to manipulate the Pipeline Context. There is an array of unsigned long for holding a bitmask and a "values" structure that holds data for the fields to be modified or queried. Use the **PEXSetOCCValueMask** function to set bits in the bitmask that correspond to the fields in the OCC that you wish to modify or query. If modifying fields, set the actual value for each field you are changing in the appropriate member of the `PEXOCCValues` data structure. If querying OCC values, read the values from the structure after invoking the query function.

The value mask bits are named:	`PEXOCCValues` member names
PEXOCCMDisplay	display
PEXOCCMRenderer	renderer
PEXOCCMStructure	structure
PEXOCCMReqType	req_type
PEXOCCMShapeHint	shape_hint
PEXOCCMIgnoreEdges	ignore_edges
PEXOCCMContourHint	contour_hint
PEXOCCMContoursAllOne	contours_all_one
PEXOCCMFacetAttributes	facet_attributes
PEXOCCMLineVertexAttributes	line_vertex_attributes
PEXOCCMMarkerVertexAttributes	marker_vertex_attributes
PEXOCCMSurfaceVertexAttributes	surface_vertex_attributes
PEXOCCMEdgeAttributes	edge_attributes
PEXOCCMFacetFPDataCount	facet_fp_data_count
PEXOCCMLineVertexFPDataCount	line_vertex_fp_data_count
PEXOCCMMarkerVertexFPDataCount	marker_vertex_fp_data_count
PEXOCCMSurfaceVertexFPDataCount	surface_vertex_fp_data_count
PEXOCCMColorType	color_type
PEXOCCMEncodingState	encoding_state
PEXOCCMDataModel	data_model
PEXOCCMDataModelSpecs	data_model_specs
PEXOCCMAll5_2	all members defined for PEXlib 5.2

PEXChangeOCC - Change the Values in an OC Context

Synopsis

void **PEXChangeOCC**(`PEXOCC` *context*, unsigned long **mask*, `PEXOCCValues` **values*)

Arguments

context	A `PEXOCC` returned by a successful **PEXCreateOCC** call.
mask	A pointer to an array of one unsigned long indicating which values provided in the `PEXOCCValues` data structure are to be used to modify the specified OC Context.
values	The address of the `PEXOCCValues` data structure containing the values to be used to modify the OC Context [§2.21.2, page 49].

Returns

None

Description

Invoke **PEXChangeOCC** to modify the values in a PEX OC Context.

If the *mask* contains all zero bits, then this function does not make any changes. Otherwise, for each defined bit that is one, this function copies the corresponding value from the *values* argument to the *context*.

Build the *mask* with the **PEXSetOCCValueMask** function (page 153).

Errors

This function ignores any undefined bits in the mask field.

Errors in setting the OC Context will not be apparent until the application actually uses the OCC as an argument in an OC function invocation.

See Also

PEXCopyOCCValues, PEXCreateOCC, **PEXGetOCCValues**, **PEXSetOCCValueMask**

PEXCopyOCCValues- Copy the Values from One OCC to Another

Synopsis

void **PEXCopyOCCValues**(`PEXOCC` *src*, unsigned long **mask*, `PEXOCC` *dest*)

Arguments

src	A `PEXOCC` returned by a successful **PEXCreateOCC** call.
mask	A pointer to an array of one unsigned long indicating which values in the `PEXOCCValues` data structure the function should copy from *src* to *dest*.
dest	A `PEXOCC` returned from a successful call to **PEXCreateOCC**.

Returns

None.

Description

Invoke **PEXCopyOCCValues** to copy the values from one PEX OC Context to another PEX OC Context.

If the *mask* contains all zero bits, then this function does not copy any values and the *dest* OC Context remains unchanged. Otherwise, for each defined bit that is one, this function copies the corresponding value from the *src* (source) context to the corresponding value in the *dest* (target) context.

Build the *mask* with the **PEXSetOCCValueMask** function (page 153).

Errors

This function ignores any undefined bits in the mask field.

See Also

PEXChangeOCC, **PEXCreateOCC**, **PEXGetOCCValues**, **PEXSetOCCValueMask**

PEXCreateOCC - Create an OC Context

Synopsis

`PEXOCC` **PEXCreateOCC**(unsigned long **mask*, `PEXOCCValues` **values*)

Arguments

mask	A pointer to an array of one unsigned long indicating which values provided in the `PEXOCCValues` data structure are to be used to modify the OC Context that is created and returned by this function.
values	The address of the `PEXOCCValues` data structure containing the values to be used to modify the OC Context [§2.21.2, page 49].

Returns

`PEXOCC` - a pointer to an opaque data structure that an application can use in subsequent OC Context manipulation functions and in OC Context attribute and primitive functions.

Description

Invoke **PEXCreateOCC** to create and return a PEX OC Context.

If the *mask* contains all zero bits, then this function sets all of the values in the context to the default values, as specified in the introduction to this section. Otherwise, for each defined bit that is one, this function copies the corresponding value from the *values* argument to the OC context.

Build the *mask* with the **PEXSetOCCValueMask** function (page 153).

Invoke the **PEXFreeOCC** function to deallocate the memory associated with the OC Context.

Errors

This function returns a NULL pointer in the case of a memory allocation error.

This function ignores undefined bits in the mask field.

Errors in setting the OC Context will not be apparent until the application actually uses the OCC as an argument in an OC function invocation.

See Also

PEXChangeOCC, PEXCopyOCCValues, PEXFreeOCC, PEXGetOCCValues, **PEXSetOCCValueMask**

PEXFreeOCC - Free an OC Context

Synopsis

void **PEXFreeOCC**(`PEXOCC` *context*)

Arguments

context	A `PEXOCC` returned by a successful **PEXCreateOCC** call.

Returns

None.

Description

Invoke **PEXFreeOCC** to deallocate the storage associated with a PEX OC Context.

After your application calls this function, the `PEXOCC` specified by the *context* argument is no longer valid for use in any OCC function.

Errors

None.

See Also

PEXChangeOCC, **PEXCopyOCCValues**, **PEXCreateOCC**, **PEXGetOCCValues**

PEXGetOCCValues- Retrieve the Values in an OC Context

Synopsis

void **PEXGetOCCValues**(`PEXOCC` *context*, unsigned long **mask*, `PEXOCCValues` **values*)

Arguments

context	A `PEXOCC` returned by a successful **PEXCreateOCC** call.
mask	A pointer to an array of one unsigned long indicating which values in the `PEXOCCValues` data structure the function should return to the caller.
values	The address of the `PEXOCCValues` data structure where you want the function to place the requested values [§2.21.2, page 49].

Returns

None.

Description

Invoke **PEXGetOCCValues** to get the values from a PEX OC Context.

If the *mask* contains all zero bits, then this function does not return any values nor does it alter any members of the *values* argument. Otherwise, for each defined bit that is one, this function copies the corresponding value from the *context* to the corresponding member in the *values* argument. All other members of the *values* structure are not altered by this function.

Build the *mask* with the **PEXSetOCCValueMask** function (page 153).

Errors

This function ignores any undefined bits in the mask field.

See Also

PEXChangeOCC, **PEXCopyOCCValues**, **PEXCreateOCC**, **PEXSetOCCValueMask**

PEXSetOCC*- Convenience Functions for Setting the OCC

Synopsis

void **PEXSetOCCDisplay**(`PEXOCC` *context*, Display **display*)
void **PEXSetOCCRenderer**(`PEXOCC` *context*, `PEXRenderer` *renderer*)
void **PEXSetOCCStructure**(`PEXOCC` *context*, `PEXStructure` *structure*)
void **PEXSetOCCReqType**(`PEXOCC` *context*, `PEXOCRequestType` *req_type*)
void **PEXSetOCCShapeHint**(`PEXOCC` *context*, int *shape_hint*)
void **PEXSetOCCIgnoreEdges**(`PEXOCC` *context*, int *ignore_edges*)
void **PEXSetOCCContourHint**(`PEXOCC` *context*, int *contour_hint*)
void **PEXSetOCCContoursAllOne**(`PEXOCC` *context*, int *contours_all_one*)
void **PEXSetOCCFacetAttributes**(`PEXOCC` *context*, unsigned int *facet_attributes*)
void **PEXSetOCCLineVertexAttributes**(`PEXOCC` *context***,** unsigned int *line_vertex_attributes*)
void **PEXSetOCCMarkerVertexAttributes**(`PEXOCC` *context***,** unsigned int *marker_vertex_attributes*)
void **PEXSetOCCSurfaceVertexAttributes**(`PEXOCC` *context***,** unsigned int *surface_vertex_attributes*)
void **PEXSetOCCEdgeAttributes**(`PEXOCC` *context*, unsigned int *edge_attributes*)
void **PEXSetOCCFacetFPDataCount**(`PEXOCC` *context*, unsigned int *facet_fp_data_count*)
void **PEXSetOCCLineVertexFPDataCount**(`PEXOCC` *context*, unsigned int *line_vertex_fp_data_count*)
void **PEXSetOCCMarkerVertexFPDataCount**(`PEXOCC` *context*, unsigned int
marker_vertex_fp_data_count)
void **PEXSetOCCSurfaceVertexFPDataCount**(`PEXOCC` *context*, unsigned int
surface_vertex_fp_data_count)
void **PEXSetOCCColorType**(`PEXOCC` *context*, int *color_type*)
void **PEXSetOCCEncodingState**(`PEXOCC` *context*, char **encoding_state*)
void **PEXSetOCCDataModel**(`PEXOCC` *context*, int *data_model*)
void **PEXSetOCCDataModelSpecsStride**(`PEXOCC` *context*, `PEXOCCStrideData` **stride*)
void **PEXSetOCCDataModelSpecsUnpacked**(`PEXOCC` *context*, `PEXOCCUnpackedData` **unpacked*)
void **PEXSetOCCAll5_2**(`PEXOCC` *context*, `PEXOCCValues` **all5_2*)

void **PEXSetOCCRequestTarget**(`PEXOCC` *context*, XID *id*, `PEXOCRequestType` *req_type*)

Arguments

context	A `PEXOCC` returned by a successful **PEXCreateOCC** call.
2nd Parameter	See the synopsis and the `PEXOCCValues` data structure [§2.21.2, page 49].

Returns

None.

Description

Invoke one of these **PEXSetOCC** functions to set a single PEX OC Context value.

PEXlib defines a function for each OCC value. Invoke the appropriate function to set the corresponding OCC value. The second argument always contains the data to copy into the OCC value.

Invoke the **PEXSetOCCAll5_2** function to set every member in the OCC. In this case, the second parameter is the address of an entire `PEXOCCValues` structure.

Invoke the **PEXSetOCCRequestTarget** function to change the *req_type* member in the OCC and either the *renderer* member (if the *req_type* is **PEXOCRender** or **PEXOCRenderSingle**) or the *structure* member (if the *req_type* is **PEXOCStore** or **PEXOCStoreSingle**).

Errors

Errors in setting the OC Context will not be apparent until the application actually uses the OCC as an argument in an OC function invocation.

See Also

PEXChangeOCC, **PEXCopyOCCValues**, **PEXCreateOCC**, **PEXGetOCCValues**, **PEXSetOCCValueMask**

PEXSetOCCValueMask - Setup OC Context Value Mask

Synopsis

void **PEXSetOCCValueMask**(unsigned long **mask*, unsigned int *attr*)

Arguments

mask	The address of the value mask - a single unsigned long.
attr	A single OC Context Value bitmask constant.

Description

Invoke the **PEXSetOCCValueMask** function to set up the bitmask for the OC Context values. See the list below for the symbols to use in creating the bitmask.

This function only sets the bit corresponding to the *attr* in the *mask*; all other bits in the *mask* remain unchanged. To set multiple bits in the *mask*, invoke this function multiple times using different values in the *attr* argument.

OCC Value Bitmask Names

The bits for the *mask* are named:	`PEXOCCValues` member names
PEXOCCMDisplay	display
PEXOCCMRenderer	renderer
PEXOCCMStructure	structure
PEXOCCMReqType	req_type
PEXOCCMShapeHint	shape_hint
PEXOCCMIgnoreEdges	ignore_edges
PEXOCCMContourHint	contour_hint
PEXOCCMContoursAllOne	contours_all_one
PEXOCCMFacetAttributes	facet_attributes
PEXOCCMLineVertexAttributes	line_vertex_attributes
PEXOCCMMarkerVertexAttributes	marker_vertex_attributes
PEXOCCMSurfaceVertexAttributes	surface_vertex_attributes
PEXOCCMEdgeAttributes	edge_attributes
PEXOCCMFacetFPDataCount	facet_fp_data_count
PEXOCCMLineVertexFPDataCount	line_vertex_fp_data_count
PEXOCCMMarkerVertexFPDataCount	marker_vertex_fp_data_count
PEXOCCMSurfaceVertexFPDataCount	surface_vertex_fp_data_count
PEXOCCMColorType	color_type
PEXOCCMEncodingState	encoding_state
PEXOCCMDataModel	data_model
PEXOCCMDataModelSpecs	data_model_specs
PEXOCCMAll5_2	all members defined for PEXlib 5.2

Errors

None.

See Also

PEXChangeOCC, **PEXCopyOCCValues**, **PEXCreateOCC**, **PEXGetOCCValues**

Chapter 6

Primitives and Attribute Binding

6.1. 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, arcs, non-uniform B-spline curves, fill areas, fill area sets, triangle primitives, quadrilateral meshes, arc areas, cones, non-uniform B-spline surfaces, cell arrays and a generalized drawing primitive (or GDP). Primitives do not alter the pipeline state; instead, the pipeline state determines their exact appearance.

Output commands which alter the rendering pipeline state are called *output primitive attributes* or simply *attributes*. Attributes, in themselves, do not produce visible output. Attributes work in conjunction with the pipeline. The pipeline context resource provides the initial renderer or PHIGS workstation pipeline state. If the application does not specify the pipeline context resource when it creates the renderer or PHIGS workstation resource, then PEXlib uses the default values. After the renderer or PHIGS workstation has obtained its initial pipeline state, only by invoking the output primitive attribute functions described in this chapter, can your application modify the pipeline attributes. You cannot directly query the current pipeline state.

Each output primitive has one or more sets of attributes applied to it. This section defines the output primitive and the attribute sets that affect them. This section lists the attribute output command (marked with "OC") and renderer attribute (marked with "RA") members of each attribute set. See the pages referred to below for the definition of each output primitive, attribute output command, and renderer attribute.

Many attributes are grouped into sets called *bundles*, which enable you to apply several attributes to a specific primitive type simultaneously. Use Attribute Source Flags (ASF's) to determine if a primitive's attribute is derived from the current bundle, as stored in a bundle table, or is derived from the individual pipeline state attribute. Attributes you can bundle are noted below with "(ASF)" to indicate that these attributes may come from one of two places, as determined by the flags.

See *"Output Attribute Functions" on page 261* for more detailed information about attributes.

6.1.1. Global Attributes

Attribute output commands that affect all output primitives are:

Add Names to Name Set OC	"PEXOCCAddToNameSet - Add Names to Name Set (5.0)" on page 262
Alpha Blending Function OC	"PEXOCCSetAlphaBlend - Set Alpha Blend Function (5.2)" on page 265
Background Color RA	"Background Color" on page 511
Clear Alpha RA	"Clear Alpha" on page 511
Clear Image RA	"Clear Image" on page 511
Clear Z-buffer RA	"Clear Z-buffer" on page 511
Clip List RA	"Clip List" on page 512
Color Approximation Index OC	"PEXOCCSetColorApproxIndex - Set Color Approximation Index (5.0)" on page 304

Color Approximation Table RA	"Color Approximation Table" on page 442 and "Color Approximation Table" on page 512
Color Table RA	"Color Table" on page 441 and "Color Table" on page 512
Depth Cue Index OC	"PEXOCCSetDepthCueIndex - Set Depth Cue Index (5.0)" on page 310
Depth Cue Table RA	"Depth Cue Table" on page 446 and "Depth Cue Table" on page 512
Drawing Function OC	"PEXOCCSetDrawingFunction - Set Drawing Function (5.2)" on page 311
Echo Mode RA	"Echo Mode" on page 513
Global Transformation OC (2D/3D)	"PEXOCCSetGlobalTransform - Set Global Transformation 3D (5.0)" on page 316
Highlight Exclusion Name Set RA	"Highlight Exclusion Name Set" on page 513
Highlight Inclusion Name Set RA	"Highlight Inclusion Name Set" on page 513
Highlight Index OC	"PEXOCCSetHighlightIndex - Set Highlight Index (5.2)" on page 322
Highlight Table RA	"Highlight Table" on page 449 and "Highlight Table" on page 513
HLHSR Identifier OC	"PEXOCCSetHLHSRID - Set Hidden-Line Hidden-Surface Removal Identifier (5.0)" on page 323
HLHSR Mode RA	"HLHSR Mode" on page 513
Individual ASF OC	"PEXOCCSetIndividualASF - Set Individual Aspect Source Flags (5.0)" on page 325
Invisibility Exclusion Name Set RA	"Invisibility Exclusion Name Set" on page 514
Invisibility Inclusion Name Set RA	"Invisibility Inclusion Name Set" on page 514
Local Transformation OC (2D/3D)	"PEXOCCSetLocalTransform - Set Local Transformation 3D (5.0)" on page 348
Model Clip Indicator OC	"PEXOCCSetModelClipFlag - Set Model Clipping Flag (5.0)" on page 359
Model Clip Volume OC (2D/3D)	"PEXOCCSetModelClipVolume - Set Model Clip Volume 3D (5.0)" on page 360
NPC Subvolume RA	"NPC Subvolume" on page 515
Pick Identifier OC	"PEXOCCSetPickID - Set Pick Identifier (5.0)" on page 372
Pipeline Context RA	"Pipeline Context" on page 515
Pixel Color Mode RA	"Pixel Color Mode" on page 515
Plane Mask OC	"PEXOCCSetPlaneMask - Set Plane Mask (5.2)" on page 373
Primitive Antialiasing OC	"PEXOCCSetPrimitiveAntialiasing - Set Primitive Antialiasing (5.2)" on page 376

Remove Names from Name Set OC	“PEXOCCRemoveFromNameSet - Remove Names from Name Set (5.0)” on page 263
Rendering State RA	“Renderer State” on page 516
Rendering Color Model OC	“PEXOCCSetRenderingColorModel - Set Rendering Color Model (5.0)” on page 383
Restore Model Clipping Volume OC	“PEXOCCRestoreModelClipVolume - Restore Model Clip Volume (5.0)” on page 264
Transparency Method RA	“Transparency Method” on page 517
Transparency Thresholds RA	“Transparency Thresholds” on page 518
View Index OC	“PEXOCCSetViewIndex - Set View Index (5.0)” on page 418
View Table RA	“View Table” on page 472
Viewport RA	“Viewport” on page 518
Z-Buffer Test OC	“PEXOCCSetZBufferTest - Set Z Buffer Test Criteria (5.2)” on page 419

6.1.2. Annotation Text Attributes

Annotation text primitives and text primitives are the same *except* that annotation primitives must appear parallel to the viewing surface. Text primitives share some attributes and duplicate others to provide independent control over annotation text appearance. See “Text Attributes” on page 163 for information about text primitives and attributes.

The annotation attributes and text attributes that affect annotation text primitives are:

Annotation Text Alignment OC	“PEXOCCSetATextAlignment - Set Annotation Text Alignment (5.0)” on page 271
Annotation Text Height OC	“PEXOCCSetATextHeight - Set Annotation Text Height (5.0)” on page 273
Annotation Text Path OC	“PEXOCCSetATextPath - Set Annotation Text Path (5.0)” on page 274
Annotation Text Style OC	“PEXOCCSetATextStyle - Set Annotation Text Style (5.0)” on page 276
Annotation Text Up Vector OC	“PEXOCCSetATextUpVector - Set Annotation Text Up Vector (5.0)” on page 277
Text Bundle Index OC	“PEXOCCSetTextBundleIndex - Set Text Bundle Index (5.0)” on page 403
Character Expansion OC (ASF)	“PEXOCCSetCharExpansion - Set Character Expansion Factor (5.0)” on page 299
Character Spacing OC (ASF)	“PEXOCCSetCharSpacing - Set Character Spacing (5.0)” on page 302
Text Color OC (ASF)	“PEXOCCSetTextColor - Set Text Color (5.0)” on page 404

Text Color Index OC (ASF)	"PEXOCCSetTextColorIndex - Set Text Color Index (5.0)" on page 406
Text Font Index OC (ASF)	"PEXOCCSetTextFontIndex - Set Text Font Index (5.0)" on page 407
Text Precision OC (ASF)	"PEXOCCSetTextPrecision - Set Text Precision (5.0)" on page 410
Text Bundle Table RA	"Text Bundle Table" on page 458
Text Font Table RA	"Text Font Table" on page 459

The PEXlib annotation text primitives affected by the above attributes include:

Annotation Text 3D	"PEXOCCAnnotationText - 3D Annotation Text Primitive (5.0)" on page 174
Annotation Text 2D	"PEXOCCAnnotationText2D - 2D Annotation Text Primitive (5.0)" on page 176
Encoded Annotation Text 2D	"PEXOCCEncodedAnnoText - Encoded 3D Annotation Text Primitive (5.0)" on page 191
Encoded Annotation Text 2D	"PEXOCCEncodedAnnoText2D - Encoded 2D Annotation Text Primitive (5.0)" on page 193

See the pages referenced above for information about each of the annotation text primitives and the attributes that affect them.

Clipping of Annotation Text: Annotation text primitives are not displayed if their origin is outside of the currently defined clipping volume. If the origin is not clipped, then the string is clipped according to the text clipping rules (see "Text Attributes" on page 163) and origin-offset connection line enabled by the annotation text style attribute is clipped according to the line clipping rules (see "Lines - Shading Attributes" on page 162). Model clipping is considered when it is determined whether the origin is clipped, but model clipping is not applied to the string or the connection line.

HLHSR (Hidden Line/ Hidden Surface Removal) of Text: The treatment of annotation text strings during hidden surface computations depends on the HLHSR (Hidden Line/Hidden Surface Removal) mode you select 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 solely from the 3D positions of each string.

Text Bundle Index and Aspect Source Flag: Whenever you render an annotation text primitive, each of the text attributes is obtained directly from the current pipeline state or indirectly from a text bundle lookup table. Your application determines the source for the attribute by the current settings of the corresponding text aspect source flag or ASF's. If your application sets the attribute's ASF to the value **PEXIndividual**, then your application obtains the attribute value directly from the current pipeline state. If your application sets an attribute's ASF to the value **PEXBundled**, then your application obtains the attribute value from the current text bundle table. The text bundle index attribute indicates the text bundle table entry to be used. The ASF 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)

6.1.3. Curve Attributes

The attributes that affect only curved line and curved surface primitives are:

Line Bundle Index OC	"PEXOCCSetLineBundleIndex - Set Line Bundle Index (5.0)" on page 337

Curve Approximation Method OC (ASF)	"PEXOCCSetCurveApprox - Set Curve Approximation Method (5.0)" on page 305
Line Bundle Table RA	"Line Bundle Table" on page 454

The PEXlib primitives affected by the above attributes include:

Arc Areas	"PEXOCCArcAreas - Arc Area Primitive (5.2)" on page 177
Arcs	"PEXOCCArcs - Arc Primitive (5.2)" on page 181
Cones	"PEXOCCCones - Cone Primitive (5.2)" on page 188
Non-uniform B-spline Curve	"PEXOCCNURBCurve - Non-Uniform Rational B-spline Curve Primitive (5.0)" on page 216
Spheres	"PEXOCCSpheres - Sphere Primitive (5.2)" on page 230

See the pages referenced above for information about each of the curve primitives and the attributes that affect them.

Curve primitives use all of the attributes of lines ("Line Attributes" on page 161).

6.1.4. Data Mapping Attributes

The data mapping attributes that affect primitives using vertex and/or facet data are:

Data Mapping Index OC[a]	"PEXOCCSetDataMappingIndex - Set Data Mapping Index (5.2)" on page 307
Data Mapping Method OC (ASF)	"PEXOCCSetDataMappingMethod - Set Data Mapping Method (5.2)" on page 308
Back-facing Data Mapping Index OC	"PEXOCCSetBFDataMappingIndex - Set Back-Facing Data Mapping Index (5.2)" on page 278
Back-facing Data Mapping Method OC (ASF)	"PEXOCCSetBFDataMappingMethod - Set Back-Facing Data Mapping Method (5.2)" on page 279
Data Mapping Table RA	"Data Mapping Table" on page 445 and "Data Mapping Table" on page 512

a. Both the data mapping index and back-facing data mapping index select from the data mapping bundle table.

The PEXlib primitives affected by the above data mapping attributes are:

Arc Areas	"PEXOCCArcAreas - Arc Area Primitive (5.2)" on page 177
Cones	"PEXOCCCones - Cone Primitive (5.2)" on page 188
Fill Area Set	"PEXOCCFillAreaSet - Fill Area Set Primitive (5.0)" on page 204
Indexed Fill Area Sets	"PEXOCCIndexedFillAreaSets - Indexed Fill Area Sets (SOFAS) Primitive (5.0)" on page 208
Indexed Triangles	"PEXOCCIndexedTriangles - Indexed Triangle Primitive (5.2)" on page 211

PolyTriangle	“PEXOCCPolyTriangle - Generalized Triangle Primitive (5.2)” on page 225
Quadrilateral Mesh	“PEXOCCQuadrilateralMesh - Quadrilateral Mesh Primitive (5.0)” on page 228
Spheres	“PEXOCCSpheres - Sphere Primitive (5.2)” on page 230
Triangle Fan	“PEXOCCTriangleFan - Triangle Fan Primitive (5.2)” on page 236
Triangles	“PEXOCCTriangles - Independent Triangles Primitive (5.2)” on page 238
Triangle Strip	“PEXOCCTriangleStrip - Triangle Strip Primitive (5.0)” on page 240

See the pages referenced above for information about each of the data primitives and the attributes that affect them.

6.1.5. Edge Attributes

The edge attributes include:

Edge Bundle Table RA	“Edge Bundle Table” on page 447 and “Edge Bundle Table” on page 513
Surface Edge Bundle Index OC	“PEXOCCSetEdgeBundleIndex - Set Edge Bundle Index (5.0)” on page 313
Surface Edge Color OC (ASF)	“PEXOCCSetSurfaceEdgeColor - Set Surface Edge Color (5.0)” on page 393
Surface Edge Color Index OC (ASF)	“PEXOCCSetSurfaceEdgeColorIndex - Set Surface Edge Color Index (5.0)” on page 394
Surface Edge Flag OC (ASF)	“PEXOCCSetSurfaceEdgeFlag - Set Surface Edge Flag (5.0)” on page 395
Surface Edge Type OC (ASF)	“PEXOCCSetSurfaceEdgeType - Set Surface Edge Type (5.0)” on page 396
Surface Edge Width OC (ASF)	“PEXOCCSetSurfaceEdgeWidth - Set Surface Edge Width (5.0)” on page 398

The PEXlib primitives affected by the above edge attributes include:

Arc Areas	“PEXOCCArcAreas - Arc Area Primitive (5.2)” on page 177
Fill Area Set	“PEXOCCFillAreaSet - Fill Area Set Primitive (5.0)” on page 204
Indexed Fill Area Sets	“PEXOCCIndexedFillAreaSets - Indexed Fill Area Sets (SOFAS) Primitive (5.0)” on page 208
Indexed Triangles	“PEXOCCIndexedTriangles - Indexed Triangle Primitive (5.2)” on page 211
Non-uniform B-spline Surface	“PEXOCCNURBSurface - Non-Uniform Rational B-spline Surface Primitive (5.0)” on page 218
PolyTriangle	“PEXOCCPolyTriangle - Generalized Triangle Primitive (5.2)” on page 225

Quadrilateral Mesh	"PEXOCCQuadrilateralMesh - Quadrilateral Mesh Primitive (5.0)" on page 228
Triangle Fan	"PEXOCCTriangleFan - Triangle Fan Primitive (5.2)" on page 236
Triangles	"PEXOCCTriangles - Independent Triangles Primitive (5.2)" on page 238
Triangle Strip	"PEXOCCTriangleStrip - Triangle Strip Primitive (5.0)" on page 240

See the pages referenced above for information about each of the primitives and the edge attributes that affect them.

6.1.6. Line Attributes

The attributes that affect line primitives are:

Line Bundle Index OC	"PEXOCCSetLineBundleIndex - Set Line Bundle Index (5.0)" on page 337
Line Cap Style OC (ASF)	"PEXOCCSetLineCapStyle - Set Line Cap Style (5.2)" on page 338
Line Color OC (ASF)	"PEXOCCSetLineColor - Set Line Color (5.0)" on page 339
Line Color Index OC (ASF)	"PEXOCCSetLineColorIndex - Set Line Color Index (5.0)" on page 341
Line Join Style OC (ASF)	"PEXOCCSetLineJoinStyle - Set Line Join Style (5.2)" on page 342
Line Type OC (ASF)	"PEXOCCSetLineType - Set Line Type (5.0)" on page 345
Line Width OC (ASF)	"PEXOCCSetLineWidth - Set Line Width (5.0)" on page 347
Line Bundle Table RA	"Line Bundle Table" on page 454 and "Line Bundle Table" on page 514
Line Miter Limit OC	"PEXOCCSetLineMiterLimit - Set Line Miter Limit (5.2)" on page 344

The PEXlib line primitives affected by the above line attributes include:

Arcs	"PEXOCCArcs - Arc Primitive (5.2)" on page 181
Non-uniform B-spline Curve	"PEXOCCNURBCurve - Non-Uniform Rational B-spline Curve Primitive (5.0)" on page 216
Polyline	"PEXOCCPolyline - Polyline Primitive (5.0)" on page 221
Polylines	"PEXOCCPolylines - Polylines Primitive (5.0)" on page 223

See the pages referenced above for information about each of the line primitives and the attributes that affect them.

Clipping of Lines: Line primitives are not displayed if you define them outside of the currently defined clipping volume. Lines which cross the clipping volume are clipped and only portions inside of the clipping volume are displayed.

Line Bundle Index and Aspect Source Flag: Whenever you render a line primitive, each of the line attributes is obtained directly from the current pipeline state or indirectly from a line bundle lookup table. Your application determines the source for the attribute by the current settings of the polyline aspect source flag or ASF's. If your application sets the attributes's ASF to the value **PEXIndividual**, then your application obtains the attribute value directly from the current pipeline state. If your application sets an attribute's ASF to the value **PEXBundled**, then your application obtains the attribute value from the current line bundle table.

6.1.7. Lines - Shading Attributes

The following attributes apply only to shaded lines:

Line Bundle Index OC	“PEXOCCSetLineBundleIndex - Set Line Bundle Index (5.0)” on page 337
Polyline Interpolation Method OC (ASF)	“PEXOCCSetPolylineInterpMethod - Set Polyline Interpolation Method (5.0)” on page 374
Line Bundle Table RA	“Line Bundle Table” on page 454 and “Line Bundle Table” on page 514

The PEXlib line primitives affected by the above line shading attributes include:

Arcs	“PEXOCCArcs - Arc Primitive (5.2)” on page 181
Polyline	“PEXOCCPolyline - Polyline Primitive (5.0)” on page 221
Polylines	“PEXOCCPolylines - Polylines Primitive (5.0)” on page 223

See the pages referenced above for information about each of the shaded line primitives and the attributes that affect them.

6.1.8. Marker Attributes

A marker is a two-dimensional glyph, or symbol, representing a location in three-dimensional space. Use modeling coordinates to specify the marker’s position. As with other output primitive coordinates, PEXlib transforms the position within the rendering pipeline. When your application enables depth-cueing, a marker’s color is affected by the depth-cueing computation. However, a marker has no surface from which to reflect light, therefore, the light-source illumination stage of the rendering pipeline has no effect on markers.

The attributes that affect marker primitives are:

Marker Bundle Index OC	“PEXOCCSetMarkerBundleIndex - Set Marker Bundle Index (5.0)” on page 352
Marker Color OC (ASF)	“PEXOCCSetMarkerColor - Set Marker Color (5.0)” on page 353
Marker Color Index OC (ASF)	“PEXOCCSetMarkerColorIndex - Set Marker Color Index (5.0)” on page 355
Marker Scale Factor OC (ASF)	“PEXOCCSetMarkerScale - Set Marker Scale (5.0)” on page 356
Marker Type OC (ASF)	“PEXOCCSetMarkerType - Set Marker Type (5.0)” on page 357
Marker Bundle Table RA	“Marker Bundle Table” on page 455 and “Marker Bundle Table” on page 514

The PEXlib marker primitives affected by the above attributes include:

Marker 3D	“PEXOCCMarkers - 3D Markers Primitive (5.0)” on page 214

See the pages referenced above for information about each of the marker primitives and the attributes that affect them.

Clipping of Markers: Clipping is performed according to the marker position. If your application positions the marker outside of the clipping volume, then the server does not render any portion of the marker glyph. Clipping of portions of marker glyphs that extend beyond the clipping volume is implementation-dependent.

HLHSR (Hidden Line/ Hidden Surface Removal) of Markers: The treatment of markers during hidden surface computations depends on the HLHSR (Hidden Line/Hidden Surface Removal) mode you select 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.

Marker Bundle Index and Marker Aspect Source Flags: Whenever you render a marker primitive, each of the marker attributes is obtained directly from the current pipeline state or indirectly from a marker bundle lookup table. Your application determines the source for the attribute by the current settings of the marker aspect source flag or ASF's. If your application sets an attribute's ASF to the value **PEXIndividual**, then your application obtains the attribute value directly from the current pipeline state. If your application sets an attribute's ASF to the value **PEXBundled**, then your application obtains the attribute value from the current marker bundle lookup table. The marker bundle index attribute indicates the marker bundle table entry to be used.

6.1.9. Text Attributes

Two types of text primitives are supported: *regular text* and *annotation text*. The server renders regular text primitives in an arbitrary 3D plane, while annotation text appears parallel to the view surface. See "Annotation Text Attributes" on page 157 for annotation text primitives and attributes.

A text string is a list of characters rendered at a particular location in three-dimensional space. The server defines text positions in modeling coordinates and transforms them into screen coordinates within the rendering pipeline. When your application enables depth-cueing, the text color is affected by the depth-cueing computation. However, a text string has no surface from which to reflect light, therefore, the light-source illumination stage of the rendering pipeline has no effect on text.

The attributes that affect regular text primitives are:

Character Height OC	"PEXOCCSetCharHeight - Set Character Height (5.0)" on page 301
Character Up Vector OC (ASF)	"PEXOCCSetCharUpVector - Set Character Up Vector (5.0)" on page 303
Text Alignment OC (ASF)	"PEXOCCSetTextAlignment - Set Text Alignment (5.0)" on page 401
Text Path OC	"PEXOCCSetTextPath - Set Text Path (5.0)" on page 408
Text Bundle Index OC	"PEXOCCSetTextBundleIndex - Set Text Bundle Index (5.0)" on page 403
Character Expansion OC (ASF)	"PEXOCCSetCharExpansion - Set Character Expansion Factor (5.0)" on page 299
Character Spacing OC (ASF)	"PEXOCCSetCharSpacing - Set Character Spacing (5.0)" on page 302
Text Color OC (ASF)	"PEXOCCSetTextColor - Set Text Color (5.0)" on page 404
Text Color Index OC (ASF)	"PEXOCCSetTextColorIndex - Set Text Color Index (5.0)" on page 406
Text Font Index OC (ASF)	"PEXOCCSetTextFontIndex - Set Text Font Index (5.0)" on page 407

Text Precision OC (ASF)	"PEXOCCSetTextPrecision - Set Text Precision (5.0)" on page 410
Text Bundle Table RA	"Text Bundle Table" on page 458 and "Text Bundle Table" on page 517
Text Font Table RA	"Text Font Table" on page 459 and "Text Font Table" on page 517

The PEXlib text primitives affected by the above attributes include:

Encoded Text 3D	"PEXOCCEncodedText - Encoded 3D Text Primitive (5.0)" on page 195
Encoded Text 2D	"PEXOCCEncodedText2D - Encoded 2D Text Primitive (5.0)" on page 197
Text 3D	"PEXOCCText - 3D Text Primitive (5.0)" on page 233
Text 2D	"PEXOCCText2D - 2D Text Primitive (5.0)" on page 235

See the pages referenced above for information about each of the text primitives and the attributes that affect them.

Clipping of Text: Clipping of text strings is dependent on the text precision attribute. If the text precision is **PEXStringPrecision**, then clipping is done in an implementation-dependent way. If the text precision is **PEXCharPrecision**, then clipping is done on at least a character-by-character basis. If the text precision is **PEXStrokePrecision**, then each character is clipped to the clipping boundaries.

HLHSR (Hidden Line/ Hidden Surface Removal) of Text: The treatment of text strings during hidden surface computations depends on the HLHSR (Hidden Line/Hidden Surface Removal) mode you select 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 solely from the 3D positions of each string.

Text Bundle Index and Aspect Source Flag: Whenever you render a text primitive, each of the text attributes is obtained directly from the current pipeline state or indirectly from a text bundle lookup table. Your application determines the source for the attribute by the current settings of the corresponding text aspect source flag or ASF's. If your application sets the attributes's ASF to the value **PEXIndividual**, then your application obtains the attribute value directly from the current pipeline state. If your application sets an attribute's ASF to the value **PEXBundled**, then your application obtains the attribute value from the current text bundle table. The text bundle index attribute indicates the text bundle table entry to be used. The ASF 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)

6.1.10. Surfaces - General Attributes

The attributes that affect all surface primitives are:

Facet Culling Mode OC	"PEXOCCSetFacetCullingMode - Set Facet Culling Mode (5.0)" on page 314
Facet Distinguish Flag OC	"PEXOCCSetFacetDistinguishFlag - Set Facet Distinguish Flag (5.0)" on page 315
Interior Bundle Index OC	"PEXOCCSetInteriorBundleIndex - Set Interior Bundle Index (5.0)" on page 329
Back-facing Reflectance Bundle Index OC[a]	"PEXOCCSetBFReflectanceBundleIndex - Set Back-Facing Reflectance Bundle Index (5.2)" on page 287

Back-facing Surface Reflection Attributes OC (ASF)	"PEXOCCSetBFReflectionAttributes - Set Back-Facing Surface Reflection Attributes (5.0)" on page 288
Back-facing Surface Reflection Model OC (ASF)	"PEXOCCSetBFReflectionModel - Set Back-Facing Surface Reflection Model (5.0)" on page 290
Reflectance Bundle Index OC[b]	"PEXOCCSetReflectanceBundleIndex - Set Reflectance Bundle Index (5.2)" on page 378
Surface Reflection Attributes OC (ASF)	"PEXOCCSetReflectionAttributes - Set Surface Reflection Attributes (5.0)" on page 379
Surface Reflection Model OC (ASF)	"PEXOCCSetReflectionModel - Set Surface Reflection Model (5.0)" on page 381
Interior Bundle Table RA	"Interior Bundle Table" on page 450 and "Interior Bundle Table" on page 514
Light Source State OC	"PEXOCCSetLightSourceState - Set Light Source State (5.0)" on page 335
Light Table RA	"Light Table" on page 452 and "Light Table" on page 514
PHIGS Interior Bundle Mode RA	"PHIGS Interior Bundle Mode" on page 515
Vertex Order OC	"PEXOCCSetVertexOrder - Set Vertex Order (5.2)" on page 417

a. The renderer's PHIGS interior bundle mode determines whether the interior bundle index or reflectance bundle index selects the specified entries of the interior bundle table.
b. The renderer's PHIGS interior bundle mode determines whether the interior bundle index or back-facing reflectance bundle index selects the specified entries of the interior bundle table.

The PEXlib surface primitives affected by the above attributes include:

Arc Areas	"PEXOCCArcAreas - Arc Area Primitive (5.2)" on page 177
Cell Array 3D	"PEXOCCCellArray - 3D Cell Array Primitive (5.0)" on page 185
Cell Array 2D	"PEXOCCCellArray2D - 2D Cell Array Primitive (5.0)" on page 187
Cone Set with Data	"PEXOCCCones - Cone Primitive (5.2)" on page 188
Extended Cell Array	"PEXOCCExtendedCellArray - Extended 3D Cell Array Primitive (5.0)" on page 199
Fill Area	"PEXOCCFillArea - Fill Area Primitive (5.0)" on page 201
Fill Area Set	"PEXOCCFillAreaSet - Fill Area Set Primitive (5.0)" on page 204
Indexed Fill Area Sets	"PEXOCCIndexedFillAreaSets - Indexed Fill Area Sets (SOFAS) Primitive (5.0)" on page 208
Indexed Triangles	"PEXOCCIndexedTriangles - Indexed Triangle Primitive (5.2)" on page 211
Non-uniform B-spline Surface	"PEXOCCNURBSurface - Non-Uniform Rational B-spline Surface Primitive (5.0)" on page 218

PolyTriangle	“PEXOCCPolyTriangle - Generalized Triangle Primitive (5.2)” on page 225
Quadrilateral Mesh with Data	“PEXOCCQuadrilateralMesh - Quadrilateral Mesh Primitive (5.0)” on page 228
Spheres	“PEXOCCSpheres - Sphere Primitive (5.2)” on page 230
Triangle Fan	“PEXOCCTriangleFan - Triangle Fan Primitive (5.2)” on page 236
Triangles	“PEXOCCTriangles - Independent Triangles Primitive (5.2)” on page 238
Triangle Strip	“PEXOCCTriangleStrip - Triangle Strip Primitive (5.0)” on page 240

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

Surface Area Bundle Index and Area Source Flags: Whenever you render a surface, each of the surface area attributes is obtained directly from the current pipeline state, or indirectly from the interior bundle table. Your application determines the source for the attribute value by the current settings of the corresponding ASF’s. If your application sets an attribute’s ASF to the value **PEXIndividual**, then your application obtains the attribute value directly from the current pipeline state. If your application sets an attribute’s ASF to the value **PEXBundled**, then your application obtains the attribute value from the current interior bundle table. The interior bundle table entry to be used is indicated by the interior bundle index attribute.

Surface Area Facets and Vertices Data: Some surface primitives allow additional data for the facet and for each vertex. If your application specifies a color value for the facet, then it is taken to be the intrinsic color of the surface and the server uses it instead of the current surface color when the surface area is rendered. Vertex colors take precedence over facet colors which override the surface color attributes. A normal specified for the surface 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.

For Fill Areas, if no surface 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 surface area is degenerate (for example, does not contain three non-colinear vertices), then the resultant value of the normal is implementation-dependent.

For Arc Areas, the geometric normal is the normalized cross product of the major and minor axes(major X minor). For elliptical arc areas, the axes are explicitly defined by the primitive. For circular arc areas, the major axis is the MC space X-axis and the minor axis is the MC space Y-axis.

For Spheres, the geometric normal of each facet of a sphere is the unit vector perpendicular to the facet and in the direction of a vector from the sphere’s center through the facet.

For Cones, the geometric normal of each facet of a cone is the unit vector perpendicular to the facet and in the direction of a vector from the cone’s axis through the facet.

6.1.11. Surfaces - Parametric Attributes

The attributes that affect parametric surface primitives are:

Interior Bundle Index OC	“PEXOCCSetInteriorBundleIndex - Set Interior Bundle Index (5.0)” on page 329

Parametric Surface Bundle Index OC[a]	"PEXOCCSetParaSurfBundleIndex - Set Parametric Surface Bundle Index (5.2)" on page 363
Parametric Surface Characteristics OC (ASF)	"PEXOCCSetParaSurfCharacteristics - Set Parametric Surface Characteristics (5.0)" on page 364
Interior Bundle Table RA	"Interior Bundle Table" on page 450 and "Interior Bundle Table" on page 514
PHIGS Interior Bundle Mode RA	"PHIGS Interior Bundle Mode" on page 515

a. The renderer's PHIGS interior bundle mode determines whether the interior bundle index or parametric surface bundle index selects the indicated entries of the interior bundle table.

The PEXlib primitives affected by the above attributes include:

Non-uniform B-spline Surface	"PEXOCCNURBSurface - Non-Uniform Rational B-spline Surface Primitive (5.0)" on page 218

6.1.12. Surfaces - Styled and Shaded Attributes

The styled and shaded surface attributes are:

Interior Bundle Index OC	"PEXOCCSetInteriorBundleIndex - Set Interior Bundle Index (5.0)" on page 329
Surface Color OC (ASF)	"PEXOCCSetSurfaceColor - Set Surface Color (5.0)" on page 390
Surface Color Index OC (ASF)	"PEXOCCSetSurfaceColorIndex - Set Surface Color Index (5.0)" on page 392
Surface Interior Style OC (ASF)	"PEXOCCSetInteriorStyle - Set Surface Interior Style (5.0)" on page 331
Surface Interior Style Index OC (ASF)	"PEXOCCSetInteriorStyleIndex - Set Surface Interior Style Index (5.0)" on page 333
Surface Interpolation Method OC (ASF)	"PEXOCCSetSurfaceInterpMethod - Set Surface Interpolation Method (5.0)" on page 399
Back-facing Interior Bundle Index OC[a]	"PEXOCCSetBFInteriorBundleIndex - Set Back-Facing Interior Bundle Index (5.2)" on page 281
Back-facing Surface Color OC (ASF)	"PEXOCCSetBFSurfaceColor - Set Back-Facing Surface Color (5.0)" on page 294
Back-facing Surface Color Index OC (ASF)	"PEXOCCSetBFSurfaceColorIndex - Set Back-Facing Surface Color Index (5.0)" on page 296
Back-facing Surface Interior Style OC (ASF)	"PEXOCCSetBFInteriorStyle - Set Back-Facing Surface Interior Style (5.0)" on page 283
Back-facing Surface Interior Style Index OC (ASF)	"PEXOCCSetBFInteriorStyleIndex - Set Back-Facing Surface Interior Style Index (5.0)" on page 285
Back-facing Surface Interpolation Method OC (ASF)	"PEXOCCSetBFSurfaceInterpMethod - Set Back-Facing Surface Interpolation Method (5.0)" on page 297

Interior Bundle Table RA	"Interior Bundle Table" on page 450 and "Interior Bundle Table" on page 514
Back-facing Surface Active Textures OC	"PEXOCCSetBFSurfaceActiveTextures - Set Back-Facing Surface Active Textures (5.2)" on page 292
Pattern Reference Point OC	"PEXOCCSetPatternAttributes - Set Pattern Interior Style Attributes (5.0)" on page 367
Pattern Reference Point and Vectors OC	"PEXOCCSetPatternAttributes - Set Pattern Interior Style Attributes (5.0)" on page 367
Pattern Size OC	"PEXOCCSetPatternSize - Set Pattern Interior Style Size (5.0)" on page 370
Pattern Table RA	"Pattern Table" on page 457 and "Pattern Table" on page 515
PHIGS Interior Bundle Mode RA	"PHIGS Interior Bundle Mode" on page 515
Surface Active Textures OC	"PEXOCCSetSurfaceActiveTextures - Set Surface Active Textures (5.2)" on page 385
Texture Mapping Perspective Correction OC	"PEXOCCSetTMPerspectiveCorrection - Set Texture Mapping Perspective Correction (5.2)" on page 412
Texture Mapping Resource Hints OC	"PEXOCCSetTMResourceHints - Set Texture Mapping Resource Hints (5.2)" on page 413
Texture Mapping Sample Frequency OC	"PEXOCCSetTMSampleFrequency - Set Texture Mapping Sample Frequency (5.2)" on page 415
Texture Mapping Table RA	"Texture Mapping Table" on page 460 and "Texture Mapping Table" on page 517

a. The renderer's PHIGS interior bundle mode determines whether the interior bundle index or back-facing interior bundle index selects the indicated entries of the interior bundle table.

PEXlib primitives affected by styled and shaded attributes include:

Arc Areas	"PEXOCCArcAreas - Arc Area Primitive (5.2)" on page 177
Cone Set with Data	"PEXOCCCones - Cone Primitive (5.2)" on page 188
Fill Area	"PEXOCCFillArea - Fill Area Primitive (5.0)" on page 201
Fill Area Set	"PEXOCCFillAreaSet - Fill Area Set Primitive (5.0)" on page 204
Indexed Fill Area Sets	"PEXOCCIndexedFillAreaSets - Indexed Fill Area Sets (SOFAS) Primitive (5.0)" on page 208
Indexed Triangles	"PEXOCCIndexedTriangles - Indexed Triangle Primitive (5.2)" on page 211
Non-uniform B-spline Surface	"PEXOCCNURBSurface - Non-Uniform Rational B-spline Surface Primitive (5.0)" on page 218
PolyTriangle	"PEXOCCPolyTriangle - Generalized Triangle Primitive (5.2)" on page 225
Quadrilateral Mesh with Data	"PEXOCCQuadrilateralMesh - Quadrilateral Mesh Primitive (5.0)" on page 228

Spheres	“PEXOCCSpheres - Sphere Primitive (5.2)” on page 230
Triangle Fan	“PEXOCCTriangleFan - Triangle Fan Primitive (5.2)” on page 236
Triangles	“PEXOCCTriangles - Independent Triangles Primitive (5.2)” on page 238
Triangle Strip	“PEXOCCTriangleStrip - Triangle Strip Primitive (5.0)” on page 240

6.1.13. Vendor Specific Attributes

PEXlib provides a generalized structure Element (GSE) output command. This command allows individual implementations to extend the supported attribute set, as well as implement other extended features. There are no standard PEX GSEs. The complete interface and behavior for each specified GSE identifier should be available with the individual PEX implementations.

GSE	“PEXOCCGSE - Generalized Structure Element (5.0)” on page 430

Chapter 7

Output Primitive Functions

7.1. Output Primitive Functions

This section describes the functions you use to generate output commands for PEX graphics primitives.

Primitive functions defined in PEXlib 5.1 have two forms in PEXlib 5.2; the original PEXlib 5.1 format and a corresponding function using the OC Context format. Some primitives, like the Cell Array primitives, have OCC and non-OCC forms that are very much alike. Others, such as the Fill Area primitive, have one OCC function that serves the role of several non-OCC functions.

PEXlib 5.2 defines primitive functions for new PEX 5.2 primitives only in the OCC form.

You should avoid using the older non-OCC forms and migrate towards using the OCC forms because the OCC provides a more convenient interface and because you can only use the new 5.2 primitives with the OCC format functions. You will always be able to achieve the same functionality using one of the newer functions. For example, if you are using **PEXPolyline2D**, you should consider migrating to **PEXOCCPolyline**, specifying the **PEXGA2D** flag to inform PEXlib that you have 2D vertex data.

For PEXlib 5.2, some primitives are reorganized into new OCC functions as described in the next section, while others are simply augmented with an OCC version of the function. The functions that were reoganized have their newer forms in the front of the chapter and their older forms in the back because there is no simple relationship between the new and old forms. This relationship is specified in Table 2, “Relationship between OCC and non-OCC Primitive Functions,” on page 171. The functions that were simply augmented have both of their forms documented on the same page.

Some primitives that existed prior to PEX 5.2 have new General Attribute flags (GA flags) that are used to indicate what facet and vertex data is present. For these primitives, the lists of supported facet and vertex attribute bits are annotated with the PEX version at which they were first supported. This annotation helps you select facet or vertex data types that are supported in the PEX version that your application is intending to use.

7.2. Simplified OCC functions for primitive OC’s

Beginning with PEXlib 5.2, the output command context (OCC) makes it possible to generate primitives that use a wide variety of attributes with a smaller number of functions. The following table describes what OCC functions to use to generate the corresponding primitive in terms of the non-OCC function form.

Table 2: Relationship between OCC and non-OCC Primitive Functions

OCC Form	non-OCC Form
PEXOCCAnnotationText	PEXAnnotationText
PEXOCCAnnotationText2D	PEXAnnotationText2D
PEXOCCArcAreas	not available
PEXOCCArcs	not available
PEXOCCCellArray	PEXCellArray
PEXOCCCellArray2D	PEXCellArray2D

Table 2: Relationship between OCC and non-OCC Primitive Functions

OCC Form	non-OCC Form
PEXOCCCones	not available
PEXOCCEncodedAnnoText	PEXEncodedAnnoText
PEXOCCEncodedAnnoText2D	PEXEncodedAnnoText2D
PEXOCCEncodedText	PEXEncodedText
PEXOCCEncodedText2D	PEXEncodedText2D
PEXOCCExtendedCellArray	PEXExtendedCellArray
PEXOCCFillArea	PEXFillArea
PEXOCCFillArea	PEXFillArea2D
PEXOCCFillArea	PEXFillAreaWithData
PEXOCCFillAreaSet	PEXFillAreaSet
PEXOCCFillAreaSet	PEXFillAreaSet2D
PEXOCCFillAreaSet	PEXFillAreaSetWithData
PEXOCCGDP	PEXGDP
PEXOCCGDP	PEXGDP2D
PEXOCCIndexedFillAreaSets	PEXSetOfFillAreaSets
PEXOCCIndexedTriangles	not available
PEXOCCMarkers	PEXMarkers
PEXOCCMarkers	PEXMarkers2D
PEXOCCNURBCurve	PEXNURBCurve
PEXOCCNURBSurface	PEXNURBSurface
PEXOCCPolyline	PEXPolyline
PEXOCCPolyline	PEXPolyline2D
PEXOCCPolylines	PEXPolylineSetWithData
PEXOCCPolyTriangle	not available
PEXOCCQuadrilateralMesh	PEXQuadrilateralMesh
PEXOCCSpheres	not available
PEXOCCText	PEXText
PEXOCCText2D	PEXText2D
PEXOCCTriangleFan	not available
PEXOCCTriangles	not available

Table 2: Relationship between OCC and non-OCC Primitive Functions

OCC Form	non-OCC Form
PEXOCCTriangleStrip	PEXTriangleStrip

7.3. Generating PEX 5.1 Output Commands

PEX 5.2 introduces five primitive output commands (**PEXOCFillAreaSetWithDataFP**, **PEXOCPolylineSetWithDataFP**, **PEXOCQuadrilateralMeshFP**, **PEXOCSetOfFillAreaSetsFP**, **PEXOCTriangleStripFP**) that are similar to their 5.1 counterparts except that they support optional floating-point values in the facet and vertex data. By default, the PEXlib 5.2 OCC-style functions generate these "FP" forms when connected to a PEX 5.2 server. However, if connected to a PEX 5.1 server, or if the Alternate Protocol Level is 5.1, PEXlib generates the 5.1 version of these output commands. If PEXlib is generating a 5.1 version of the opcode and you supply data via the OCC-stlye function that is superfluous or in conflict with the capabilities of the PEX 5.1 output command, you may cause an output command error.

Note that you may also always generate PEX 5.1 output commands by using the non-OCC forms of the functions.

Also note that for many OCC functions, the implementation may generate one of several possible OC's, depending on the setting of the "GA" flags. Applications that fetch elements from a structure would need to be particularly aware of this, since any application code that analyzes structure content would need to take into account the possibility of a structure element being one out of several possible types. Each OCC function that can generate more than one OC type is documented with a list of these possible OC types.

PEXOCCAnnotationText - 3D Annotation Text Primitive (5.0)

Synopsis

void **PEXOCCAnnotationText**(`PEXOCC` *context*, `PEXCoord` **origin*, `PEXCoord` **offset*, int *length*, char **string*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
origin	The origin of the text string [§2.7.1, page 33].
offset	The relative position of the text string from the *origi*n [§2.7.1, page 33]
length	The number of bytes in the text string.
string	A pointer to the text string.

Returns

None.

Description

Invoke **PEXOCCAnnotationText** to create an annotation text primitive output command (**PEXOCAnnotationText**).

This primitive uses the first character set in the text font. Define *origin* to specify, in modeling coordinates, the position of the first character in the text string. Define *offset* as a value, in normalized projection coordinates, indicating the relative position of the text from *origin*. The point where the implementation places the text string is called the annotation point. The implementation computes the annotation point 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, this primitive obtains the attributes: text color, text precision, character expansion, character spacing, and the text font index either directly from the current text attribute values or from the entry in the text bundle lookup table specified by the current text bundle index attribute. This primitive also applies the annotation text height, annotation text path, annotation text alignment, annotation text up vector, and annotation text style attributes to the text string.

Only the depth-cueing computation affects the annotation text string's color (the reflectance stage of the rendering pipeline affects only surface primitives) and maps it to a device color.

If your application clips the origin of the primitive, then the implementation clips the entire annotation text primitive. If your application clips the primitive but not the origin, then the implementation clips the annotation text according to PEX text clipping rules. If there is a connection line, then the implementation clips the line according to polyline clipping rules, except that it ignores modeling clipping. The implementation renders the connection line according to the current set of polyline attributes.

Errors

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

See Also

For primitives similar to this one, see: **PEXOCCAnnotationText2D, PEXOCCEncodedAnnoText, PEXOCCEncodedAnnoText2D, PEXOCCEncodedText, PEXOCCEncodedText2D, PEXOCCText, PEXOCCText2D.**

For PEXlib 5.1 compatibility primitives similar to this one, see: **PEXAnnotationText**, **PEXAnnotationText2D, PEXEncodedAnnoText, PEXEncodedAnnoText2D, PEXEncodedText, PEXEncodedText2D, PEXText, PEXText2D.**

For attributes that affect this primitive, see "Annotation Text Attributes" on page 157.

PEXOCCAnnotationText2D - 2D Annotation Text Primitive (5.0)

Synopsis

void **PEXOCCAnnotationText2D**(`PEXOCC` *context*, `PEXCoord2D` **origin*, `PEXCoord2D` **offset*, int *length*, char **string*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
origin	The origin of the text string [§2.7.2, page 33].
offset	The relative position of the text string from the *origin* [§2.7.2, page 33].
length	The number of bytes in the text string.
string	A pointer to the text string.

Returns

None.

Description

Invoke **PEXOCCAnnotationText2D** to create a two-dimensional annotation text primitive output command (**PEXOCAnnotationText2D**).

This primitive functions like the **PEXAnnotationText** primitive except that you only need to specify x- and y-components for the *origin* and *offset* positions. The z-component is always assumed to be zero. This primitive is two-dimensional only in that the z-component is implied. The implementation still carries out the geometry transformations in three dimensions.

Errors

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

See Also

For primitives similar to this one, see: **PEXOCCAnnotationText, PEXOCCEncodedAnnoText, PEXOCCEncodedAnnoText2D, PEXOCCEncodedText, PEXOCCEncodedText2D, PEXOCCText, PEXOCCText2D.**

For PEXlib 5.1 compatibility primitives similar to this one, see: **PEXAnnotationText**, **PEXAnnotationText2D, PEXEncodedAnnoText, PEXEncodedAnnoText2D, PEXEncodedText, PEXEncodedText2D, PEXText, PEXText2D.**

For attributes that affect this primitive, see “Annotation Text Attributes” on page 157.

PEXOCCArcAreas - Arc Area Primitive (5.2)

Synopsis

void **PEXOCCArcAreas**(`PEXOCC` *context*, `PEXArcAreaType` *arc_type*, unsigned int *count*,
`PEXPointer` *arcs*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
color_type (OCC)	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
surface_vertex_attributes (OCC)	A mask indicating the surface vertex attributes provided. (**PEXGAColor**, **PEXGA2D**, and **PEXGAFloatData**)
surface_vertex_fp_data_count (OCC)	Number of floats provided for each vertex.
arc_type	The type of arc area requested (**PEXArcAreaCircle**, **PEXArcAreaCircularClosedChordArc**, **PEXArcAreaCircularClosedPieArc**, **PEXArcAreaEllipse**, **PEXArcAreaEllipticalClosedChordArc,** or **PEXArcAreaEllipticalClosedPieArc**) [§2.3.1, page 28].
count	The number of arc areas.
arcs	A pointer to the arc area data [§2.18.1, page 46]. See the description below for an explanation of this data. See “Facet / Vertex Data Formats” on page 137.
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCArcAreas** to create an arc-based fill areas primitive output command (**PEXOCArcAreaSetWithData**). This implementation applies the General Surface, Styled and Shaded Surface, Edge, Curve, and Data Mapping attribute sets and surface clipping to this primitive.

You can only define one type of arc area per function call. PEXlib defines the following arc-based fill areas:

PEXArcAreaCircle - specifies a curve, which is everywhere equidistant from the center vertex in the z=0 plane. A circle also includes the interior. If your application defines a three-dimensional circle, then the implementation ignores the z values. If the radius of the circle is less than or equal to zero, then the circle degrades to a dot at the center vertex.

PEXArcAreaCircularClosedChordArc -
PEXArcAreaCircularClosedPieArc - specifies a circular arc in the z=0 plane defined by the radius, a start

angle(α), and a stop angle(β) as follows:

$$\sum_{A=\alpha}^{\beta} M\cos A + m\sin A$$

where:

M = <radius, 0, 0>,
m = <0, radius, 0>

If $|\beta - \alpha| > 2\pi$, then $\beta = \alpha + 2\pi$.

If $\beta < \alpha$, then $\beta = \beta + 2\pi$.

You close the arc area by joining each endpoint by a chordal segment (**PEXArcAreaCircularClosedChordArc**) or by connecting each endpoint with a segment to the center vertex (**PEXArcAreaCircularClosedPieArc**). If your application defines a three-dimensional circular closed arc, then the implementation ignores the z value. If the radius of the circular closed arc is less than or equal to zero, then the arc degrades to a dot at the center vertex.

PEXArcAreaEllipse - specifies a curve, defined by a center vertex and major and minor axes, and its interior. The major and minor axes define a parallelogram that circumscribes the ellipse and is anchored at the center vertex of the ellipse. Two sides of the parallelogram are parallel to the major axis. The other two sides are parallel to the minor axis. The major and minor axes extend from the center of the ellipse to the midpoints of the sides of the parallelogram. The ellipse is tangent to the sides of the parallelogram at their midpoints. If the length of both of the axes is equal to zero, then the ellipse degrades to a dot at the center vertex. If the length of just one of the axes is equal to zero, or if the axes have non-zero lengths and are parallel, then the ellipse degrades in an implementation-dependent way.

PEXArcAreaEllipticalClosedChordArc -
PEXArcAreaEllipticalClosedPieArc - specifies an elliptical arc area defined by a start angle(α), and a stop angle(β) as follows:

$$\sum_{A=\alpha}^{\beta} MajorAxis\cos A + MinorAxis\sin A$$

If $|\beta - \alpha| > 2\pi$, then $\beta = \alpha + 2\pi$.

If $\beta < \alpha$, then $\beta = \beta + 2\pi$.

You close the arc area by joining each endpoint by a segment (**PEXArcAreaEllipticalClosedChordArc**) or by connecting each endpoint with a segment to the center vertex (**PEXArcAreaEllipticalClosedPieArc**). If the length of both of the axes is equal to zero, then the arc degrades to a dot at the center vertex. If the length of just one of the axes is equal to zero or if the axes have non-zero lengths and are parallel, then the elliptical closed arc degrades in an implementation-dependent way.

This primitive only supports the optional vertex attributes: **PEXGAColor, PEXGAFloatData,** and **PEXGA2D**.

Although there is no non-OCC form of this function, you may use one of the **PEXDataPacked**, **PEXDataStride**, or the **PEXDataPacked** data models to supply the arc area data via the *arcs* parameter in modeling coordinates.

Each arc area type has its own set of required vertex data, where each vertex is used to describe a complete arc area.

Table 3: Required Vertex Data for Arc Area Primitives

Arc Area Type	Required Vertex Data
PEXArcAreaCircle	coord, radius
PEXArcAreaCircularClosedChordArc	coord, radius, angles
PEXArcAreaCircularClosedPieArc	coord, radius, angles
PEXArcAreaEllipse	coord, axes
PEXArcAreaEllipticalClosedChordArc	coord, axes, angles
PEXArcAreaEllipticalClosedPieArc	coord, axes, angles

PEXlib ignores the **PEXGARadius** flag since the presence of a radius is implicit with the arc area type.

PEXlib does not supply a set of type definitions for arc area set vertex data due to the large number of combinations of vertex data possible when considering both the required and optional vertex data. Therefore, you may find it convenient to design your own data structure to pass in the *arcs* parameter when using the **PEXDataPacked** data model. You need to design the data structure with the following ordering rules in mind:

For the three circular arc area types, the required order is:

```
      PEXCoord          (or PEXCoord2D) center
      PEXColor*         one of the PEXlib color types, if provided
      float             radius
      float             start angle, if an arc
      float             end angle, if an arc
      float[n]          floating point data, if provided
```

For the three elliptical arc area types, the required order is:

```
      PEXCoord          (or PEXCoord2D) center
      PEXColor*         one of the PEXlib color types, if provided
      PEXCoord          (or PEXCoord2D) major axis
      PEXCoord          (or PEXCoord2D) minor axis
      float             start angle, if an arc
      float             end angle, if an arc
      float[n]          floating point data, if provided
```

As an example, if you wish to supply data for Closed Circular Arcs using 3D coordinates, RGB floating-point colors and three floating-point numbers (for texture data), you would create a structure like this:

```
typedef struct {
      PEXCoord          center;
      PEXColorRGB       rgb;
      float             radius;
      float             start_angle;
      float             stop_angle;
      float             texture_data[3];
} MY_PEXRGBCircularClosedArc;
```

Remember to set the correct surface vertex attributes in the OCC. In this case you would make sure that the **PEXGA2D** flag is off, the **PEXGAColor** is on and the **PEXGAFloatData** flag is on. You set the *color_type* field in the OCC to **PEXColorTypeRGB** and the number of floats in the OCC to three.

If you are using the **PEXDataStride** data model, then you set the stride and offset members in the OC Context to reflect the way the data is stored in your structured array. Pass a pointer to the start of the structured array in the *arcs* parameter of the function call. Be sure that there is a valid offset for each vertex attribute that is present.

If you are using the **PEXDataUnpacked** data model, then you would set size members in the OCC to the appropriate values and point to each of the data lists with the "PEXUnpackedVertexData Data Structure" on page 51. Again, be careful to set a size and pointer for every attribute you indicate as present in the vertex attributes.

Errors

BadPEXOutputCommand	The specified arc area type is invalid.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

For primitives similar to this one, see: **PEXOCCArcs.**

There are no PEXlib 5.1 compatibility primitives similar to this one.

For attributes that affect this primitive, see "Curve Attributes" on page 158, "Data Mapping Attributes" on page 159, "Edge Attributes" on page 160, "Surfaces - General Attributes" on page 164, "Surfaces - Styled and Shaded Attributes" on page 167.

PEXOCCArcs - Arc Primitive (5.2)

Synopsis

void **PEXOCCArcs**(`PEXOCC` *context*, `PEXArcType` *arc_type*, unsigned int *count*, `PEXPointer` *arcs*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
color_type (OCC)	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
line_vertex_attributes (OCC)	A mask indicating the vertex attributes provided (**PEXGAColor**, **PEXGA2D**, and **PEXGAFloatData**).
line_vertex_fp_data_count (OCC)	Number of floats provided for each vertex.
arc_type	The type of arc requested (**PEXArcCircle, PEXArcCircularArc, PEXArcCircularClosedChordArc, PEXArcCircularClosedPieArc, PEXArcEllipse, PEXArcEllipticalArc, PEXArcEllipticalClosedChordArc** or **PEXArcEllipticalClosedPieArc**) [§2.3.2, page 28]**.**
count	The number of arcs.
arcs	A pointer to the arc data [§2.18.1, page 46]. See the description below for an explanation of this data. See "Facet / Vertex Data Formats" on page 137.
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCArcs** to create an arc-based line primitive output command (**PEXOCArcSetWithData**). The implementation applies the Line, Shaded Line, and Curve attribute sets, and line clipping to this primitive.

You can only define one type of arc per function call. PEXlib defines the following arc-based lines:

PEXArcCircle - specifies a curve, which is everywhere equidistant from the center vertex in the z=0 plane. If your application defines a three-dimensional circle, then the implementation ignores the z values. If the radius of the circle is less than or equal to zero, then the circle degrades to a dot at the center vertex.

PEXArcCircularArc -
PEXArcCircularClosedChordArc -
PEXArcCircularClosedPieArc - specifies a circular arc in the z=0 plane defined by the radius, a start

angle(α), and a stop angle(β) as follows:

$$\sum_{A = \alpha}^{\beta} M \cos A + m \sin A$$

where:

M = <radius, 0, 0>,
m = <0, radius, 0>

If $|\beta - \alpha| > 2\pi$, then $\beta = \alpha + 2\pi$.

If $\beta < \alpha$, then $\beta = \beta + 2\pi$.

You leave the arc open by specifying the **PEXArcCircularArc** arc type or close it with a chordal segment (**PEXArcCircularClosedChordArc**) or by connecting each endpoint to the center vertex (**PEXArcCircularClosedPieArc**). If your application defines a three-dimensional circular closed arc, then the implementation ignores the z value. If the radius of the circular closed arc is less than or equal to zero, then the arc degrades to a dot at the center vertex.

PEXArcEllipse - specifies a curve, defined by a center vertex and major and minor axes. The major and minor axes define a parallelogram that circumscribes the ellipse and is anchored at the center vertex of the ellipse. Two sides of the parallelogram are parallel to the major axis. The other two sides are parallel to the minor axis. The major and minor axes extend from the center of the ellipse to the midpoints of the sides of the parallelogram. The ellipse is tangent to the sides of the parallelogram at their midpoints. If the length of both of the axes is equal to zero, then the ellipse degrades to a dot at the center vertex. If the length of just one of the axes is equal to zero, then the ellipse degrades to a segment representing the other axis. In both of these cases, PEXlib applies edge attributes. If the axes have non-zero lengths and are parallel, then the ellipse degrades in an implementation-dependent way.

PEXArcEllipticalArc -
PEXArcEllipticalClosedChordArc -
PEXArcEllipticalClosedPieArc - specifies an elliptical arc defined by a start angle(α), and a stop angle(β) as follows:

$$\sum_{A = \alpha}^{\beta} MajorAxis \cos A + MinorAxis \sin A$$

If $|\beta - \alpha| > 2\pi$, then $\beta = \alpha + 2\pi$.

If $\beta < \alpha$, then $\beta = \beta + 2\pi$.

You leave the arc open by specifying the **PEXArcEllipticalArc** arc type or close it with a segment between the two endpoints (**PEXArcEllipticalClosedChordArc**) or connect each endpoint with a segment to the center vertex (**PEXArcEllipticalClosedPieArc**). If the length of both of the axes is equal to zero, then the arc degrades to a dot at the center vertex. If the length of just one of the axes is equal to zero or if the axes have non-zero lengths and are parallel, then the arc degrades in an implementation-dependent way.

This primitive only supports the optional vertex_attributes: **PEXGAColor, PEXGAFloatData,** and **PEXGA2D**.

Although there is no non-OCC form of this function, you may use the **PEXDataPacked**, **PEXDataStride**, or the **PEXDataPacked** data models to supply the arc data via the *arcs* parameter in modeling coordinates.

Each arc type has its own set of required vertex data, where each vertex is used to describe a complete arc.

Table 4: Required Vertex Data for Arc Primitives

Arc Type	Required Vertex Data
PEXArcCircle	coord, radius
PEXArcCircularArc	coord, radius, angles
PEXArcCircularClosedChordArc	coord, radius, angles
PEXArcCircularClosedPieArc	coord, radius, angles
PEXArcEllipse	coord, axes
PEXArcEllipticalArc	coord, axes, angles
PEXArcEllipticalClosedChordArc	coord, axes, angles
PEXArcEllipticalClosedPieArc	coord, axes, angles

PEXlib ignores the **PEXGARadius** flag since the presence of a radius is implicit with the arc type.

PEXlib does not supply a set of type definitions for arc set vertex data due to the large number of combinations of vertex data possible when considering both the required and optional vertex data. Therefore, you may find it convenient to design your own data structure to pass in the *arcs* parameter when using the **PEXDataPacked** data model. You need to design the data structure with the following ordering rules in mind:

For the four circular arc types, the required order is:

```
PEXCoord          (or PEXCoord2D) center
PEXColor*         one of the PEXlib color types, if provided
float             radius
float             start angle, if an arc
float             end angle, if an arc
float[n]          floating point data, if provided
```

For the four elliptical arc area types, the required order is:

```
PEXCoord          (or PEXCoord2D) center
PEXColor*         one of the PEXlib color types, if provided
PEXCoord          (or PEXCoord2D) major axis
PEXCoord          (or PEXCoord2D) minor axis
float             start angle, if an arc
float             end angle, if an arc
float[n]          floating point data, if provided
```

As an example, if you wish to supply data for Closed Circular Arcs using 3D coordinates, RGB floating-point colors and three floating-point numbers (for texture data), you would create a structure like this:

```
typedef struct {
      PEXCoord          center;
      PEXColorRGB       rgb;
      float             radius;
      float             start_angle;
      float             stop_angle;
      float             texture_data[3];
} MY_PEXRGBCircularClosedArc;
```

Remember to set the correct line vertex attributes in the OCC. In this case you would make sure that the **PEX-GA2D** flag is off, the **PEXGAColor** is on and the **PEXGAFloatData** flag is on. You set the *color_type* field in the OCC to **PEXColorTypeRGB** and the number of floats in the OCC to three.

If you are using the **PEXDataStride** data model, then you set the stride and offset members in the OC Context to reflect the way the data is stored in your structured array. Pass a pointer to the start of the structured array in the *arcs* parameter of the function call. Be sure that there is a valid offset for each vertex attribute that is present.

If you are using the **PEXDataUnpacked** data model, then you would set size members in the OCC to the appropriate values and point to each of the data lists with the "PEXUnpackedVertexData Data Structure" on page 51. Again, be careful to set a size and pointer for every attribute you indicate as present in the vertex attributes.

Errors

BadPEXOutputCommand	The specified arc type is invalid.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

For primitives similar to this one, see: **PEXOCCArcAreas.**

There are no PEXlib 5.1 compatibility primitives similar to this one.

For attributes that affect this primitive, see "Curve Attributes" on page 158, "Line Attributes" on page 161, "Lines - Shading Attributes" on page 162

PEXOCCCellArray - 3D Cell Array Primitive (5.0)

Synopsis

void **PEXOCCCellArray**(`PEXOCC` *context*, `PEXCoord` **point1*, `PEXCoord` **point2*, `PEXCoord` **point3*, unsigned int *col_count*, unsigned int *row_count*, `PEXTableIndex` **color_indices*)

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

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
point1	The first cell array definition point [§2.7.1, page 33].
point2	The second cell array definition point [§2.7.1, page 33].
point3	The third cell array definition point [§2.7.1, page 33].
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 [§2.3.21, page 29].

Returns

None.

Description

Invoke **PEXOCCCellArray** to create a three-dimensional cell-array primitive output command (**PEXOCCellArray**).

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:

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

and a height defined by:

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

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*.

PEXlib does not support a value of zero for the column count and row count (*col_count* and *row_count* arguments respectively).

If your application does not define a color index, then the color index defaults to a value of one. If the implementation does not define color index one, then the result is implementation-dependent.

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

For primitives similar to this one, see: **PEXOCCCellArray2D**, **PEXOCCExtendedCellArray.**

For PEXlib 5.1 compatibility primitives similar to this one see: **PEXCellArray, PEXCellArray2D**, **PEXExtendedCellArray**

For attributes that affect this primitive, see "Surfaces - General Attributes" on page 164.

PEXOCCCellArray2D - 2D Cell Array Primitive (5.0)

Synopsis

void **PEXOCCCellArray2D**(`PEXOCC`*context*, `PEXCoord2D` **point1*, `PEXCoord2D` **point2*, unsigned int *col_count*, unsigned int *row_count*, `PEXTableIndex` **color_indices*)

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

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
point1	The first cell array definition point [§2.7.2, page 33].
point2	The second cell array definition point [§2.7.2, page 33].
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 [§2.3.21, page 29].

Returns

None.

Description

Invoke **PEXOCCCellArray2D** to create a two-dimensional cell array primitive output command (**PEXOC-CellArray2D**).

A 2D cell-array is a rectangle of equally-sized cells, each of which is a rectangle which has a single color. Specify this primitive as 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 the **PEXCellArray** primitive.

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

For primitives similar to this one, see: **PEXOCCCellArray**, **PEXOCCExtendedCellArray.**

For PEXlib 5.1 compatibility primitives similar to this one see: **PEXCellArray, PEXCellArray2D**, **PEXExtendedCellArray**

For attributes that affect this primitive, see “Surfaces - General Attributes” on page 164.

PEXOCCCones - Cone Primitive (5.2)

Synopsis

void **PEXOCCCones**(`PEXOCC` *context*, double *lower_radius,* double *upper_radius,* int *lower_cap,* int *upper_cap*, unsigned int *num_cones*, `PEXPointer` *cones*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
color_type (OCC)	The type of color for the cones (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
surface_vertex_attributes (OCC)	A mask indicating the vertex attributes provided (**PEXGAColor**, **PEXGARadius**, **PEXGAFloatData**).
surface_vertex_fp_data_count (OCC)	Number of floats provided for each vertex.
lower_radius	The radius of the lower end of the cone.
upper_radius	The radius of the upper end of the cone.
lower_cap	The flag to control the drawing of the lower cap of the cones (**PEXOn** or **PEXOff**).
upper_cap	The flag to control the drawing of the upper cap of the cones (**PEXOn** or **PEXOff**).
num_cones	The number of cones.
cones	A pointer to the cone vertex data. There are two vertices per cone. See the description below for details. See "Facet / Vertex Data Formats" on page 137.
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCCones** to create a cone primitive output command (**PEXOCConeSetWithData**).

A cone is defined as a three-dimensional surface with upper and lower edges defined by two circles. These circles are perpendicular to the axis which passes through their centers. The sides of the cone are defined by a segment of the shortest possible length connecting the two circles as it is swept around their circumferences. The center vertex and radius of each circle are defined in modeling coordinates.

You specify each cone with two vertices and pass the list of vertices in the argument *cones*. Therefore the number of vertices you specify must be twice the number of cones you indicate with *num_cones*. The lower vertex is the first vertex of the two that define each cone.

Use *upper_cap and lower_cap* to control the capping of the cones. If you set a cap flag to **True**, then the implementation treats the bounding circle as a fill area with interior style **PEXInteriorStyleSolid**. Otherwise, it treats the bounding circle as a fill area with interior style **PEXInteriorStyleHollow**. The cap flags apply to all the cones in the list.

Use *surface_vertex_attributes* to specify optional data for the two center vertices of each cone. The supported components of the bitmask are **PEXGAColor, PEXGARadius** and **PEXGAFloatData**. You must always include coordinate data (not 2D) in the vertex data.

If you set the **PEXGAColor** bit in the *surface_vertex_attributes* bitmask, then you supply a color with each center vertex. The colors are of the type specified by *color_type*. Otherwise, the implementation ignores *color_type*. The implementation applies to the cone the color associated with the center vertex of the lower circle. The implementation ignores the color associated with the center vertex of the upper circle.

If you set the **PEXConeRadius** bit in the *surface_vertex_attributes* bitmask, then you supply the radius for each end of the cone in each center vertex and the implementation ignores the *lower_radius* and *upper_radius* arguments. Otherwise the *lower_radius* and *upper_radius* arguments define the common radii applied to all cones in the set. If a radius is less than zero or both radii are equal to zero, then the results of rendering that cone are implementation-dependent.

If you set the **PEXGAFloatData** bit in the *surface_vertex_attributes* bitmask, then you supply extended data with each center vertex. The *surface_fp_data_count* member indicates the number of floating-point values in the LISTofFLOAT for each vertex. If you do not set the **PEXGAFloatData** bit, then the implementation ignores *surface_fp_data_count*.

PEXlib clips **PEXConeSetWithData** similar to **PEXFillArea3D** “Surfaces - General Attributes” on page 164.

PEXlib does not supply a set of type definitions for cones due to the large number of combinations of vertex data possible when considering both the required and optional vertex data. Therefore, you may find it convenient to design your own data structure to pass in the *cones* parameter when using the **PEXDataPacked** data model. You need to design the data structure with the following ordering rules in mind:

The required order is:

```
      PEXCoord          center
      PEXColor*         one of the PEXlib color types, if provided
      float             radius
      float[n]          floating point data, if provided
```

As an example, if you wish to supply data for cones using coordinates, RGB floating-point colors, and a radius, you would create a structure like this:

```
typedef struct {
      PEXCoord          center;
      PEXColorRGB       rgb;
      float             radius;
} MY_PEXRGBConeVertex;
```

Remember to set the correct surface vertex attributes in the OCC. In this case you would make sure that the **PEXGAColor** flag is on and the **PEXGARadius** flag is on. You set the *color_type* field in the OCC to **PEXColorTypeRGB**. You need to supply two vertices per cone in your list.

If you are using the **PEXDataStride** data model, then you set the stride and offset members in the OC Context to reflect the way the data is stored in your structured array. Pass a pointer to the start of the structured array in the *cones* argument of the function call. Be sure that there is a valid offset for each vertex attribute that is present.

If you are using the **PEXDataUnpacked** data model, then you would set size members in the OCC to the appropriate values and point to each of the data lists with the “PEXUnpackedVertexData Data Structure” on page 51. Again, be careful to set a size and pointer for every attribute you indicate as present in the vertex attributes.

Errors

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

See Also

For primitives similar to this one, see: **PEXOCCSpheres.**

There are no PEXlib 5.1 compatibility primitives similar to this one.

For attributes that affect this primitive, see "Curve Attributes" on page 158, "Data Mapping Attributes" on page 159, "Surfaces - General Attributes" on page 164, "Surfaces - Styled and Shaded Attributes" on page 167.

PEXOCCEncodedAnnoText - Encoded 3D Annotation Text Primitive (5.0)

Synopsis

void **PEXOCCEncodedAnnoText**(`PEXOCC` *context*, `PEXCoord` **origin*, `PEXCoord` **offset*, unsigned int *count*, `PEXEncodedTextData` **encoded_text*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
origin	The origin of the text string [§2.7.1, page 33].
offset	The relative position of the text string from the *origin* [§2.7.1, page 33].
count	The number of encoded text strings.
encoded_text	An array of encoded text strings [§2.13.1, page 37].

Returns

None.

Description

Invoke **PEXOCCEncodedAnnoText** to create an annotation text primitive output command (**PEXOCAnnotationText**).

This primitive is similar to the **PEXAnnotationText** primitive, except that you can specify multiple encoded text strings. 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 implementation uses the default font group to render the entire string. If a character set value is not available in the default font group, then the implementation renders that portion of the string in an implementation-dependent way.

The character set width indicates the width or size of characters in the strings. PEXlib defines the following character set widths: **PEXCSByte**, **PEXCSShort** and **PEXCSLong**. The PEX implementation does not interpret the encoding state, only the application uses it.

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

Errors

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

See Also

For primitives similar to this one, see:**PEXOCCAnnotationText**, **PEXOCCAnnotationText2D, PEXOCCEncodedAnnoText2D, PEXOCCEncodedText, PEXOCCEncodedText2D, PEXOCCText, PEXOCCText2D.**

For PEXlib 5.1 compatibility primitives similar to this one, see: **PEXAnnotationText**, **PEXAnnotationText2D, PEXEncodedAnnoText, PEXEncodedAnnoText2D, PEXEncodedText, PEXEncodedText2D, PEXText, PEXText2D.**

For attributes that affect this primitive, see “Annotation Text Attributes” on page 157.

PEXOCCEncodedAnnoText2D - Encoded 2D Annotation Text Primitive (5.0)

Synopsis

void **PEXOCCEncodedAnnoText2D**(PEXOCC *context*, PEXCoord2D **origin*, PEXCoord2D **offset*, unsigned int *count*, PEXEncodedTextData **encoded_text*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
origin	The origin of the text string [§2.7.2, page 33].
offset	The relative position of the text string from the *origin* [§2.7.2, page 33].
count	The number of encoded text strings.
encoded_text	An array of encoded text strings [§2.13.1, page 37].

Returns

None.

Description

Invoke **PEXOCCEncodedAnnoText2D** to create a two-dimensional annotation text primitive output command (**PEXOCAnnotationText2D**).

This primitive is similar to the **PEXAnnotationText2D** primitive, except that you can specify multiple encoded text strings. 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 implementation uses the default font group to render the entire string. If a character set value is not available in the default font group, then the implementation renders that portion of the string in an implementation-dependent way.

The character set width indicates the width or size of characters in the strings. PEXlib defines the following character set widths: **PEXCSByte**, **PEXCSShort** and **PEXCSLong**. The PEX implementation does not interpret the encoding state; only the application uses it.

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

Errors

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

See Also

For primitives similar to this one, see:**PEXOCCAnnotationText**, **PEXOCCAnnotationText2D, PEXOCCEncodedAnnoText, PEXOCCEncodedText, PEXOCCEncodedText2D, PEXOCCText, PEXOCCText2D.**

For PEXlib 5.1 compatibility primitives similar to this one, see: **PEXAnnotationText**, **PEXAnnotationText2D, PEXEncodedAnnoText, PEXEncodedAnnoText2D, PEXEncodedText, PEXEncodedText2D, PEXText, PEXText2D.**

For attributes that affect this primitive, see “Annotation Text Attributes” on page 157.

PEXOCCEncodedText - Encoded 3D Text Primitive (5.0)

Synopsis

void **PEXOCCEncodedText**(PEXOCC *context*, PEXCoord **origin*, PEXVector **vector1*, PEXVector **vector2*, unsigned int *count*, PEXEncodedTextData **encoded_text*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
origin	The origin of the text string [§2.7.1, page 33].
vector1	A vector defining the positive x-direction of the text local coordinate system [§2.31.1, page 67].
vector2	A vector defining the positive y-direction of the text local coordinate system [§2.31.1, page 67].
count	The number of encoded text strings.
encoded_text	An array of encoded text strings [§2.13.1, page 37].

Returns

None.

Description

Invoke **PEXOCCEncodedText** to create a text primitive output command (**PEXOCText**).

This primitive is similar to the **PEXText** primitive, except that you can specify multiple encoded text strings. 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 implementation uses the default font group to render the entire string. If a character set value is not available in the default font group, then the implementation renders that portion of the string in an implementation-dependent way.

The character set width indicates the width or size of characters in the strings. PEXlib defines the following character set widths: **PEXCSByte**, **PEXCSShort** and **PEXCSLong**. The implementation does not interpret the encoding state; only the application uses it.

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

Errors

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

See Also

For primitives similar to this one, see:**PEXOCCAnnotationText**, **PEXOCCAnnotationText2D, PEXOCCEncodedAnnoText, PEXOCCEncodedAnnoText2D, PEXOCCEncodedText2D, PEXOCCText, PEXOCCText2D.**

For PEXlib 5.1 compatibility primitives similar to this one, see: **PEXAnnotationText**, **PEXAnnotationText2D, PEXEncodedAnnoText, PEXEncodedAnnoText2D, PEXEncodedText, PEXEncodedText2D, PEXText, PEXText2D.**

For attributes that affect this primitive, see "Text Attributes" on page 163.

PEXOCCEncodedText2D - Encoded 2D Text Primitive (5.0)

Synopsis

void **PEXOCCEncodedText2D**(`PEXOCC` *context*, `PEXCoord2D` **origin*, unsigned int *count*,
`PEXEncodedTextData` **encoded_text*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
origin	The origin of the text string [§2.7.2, page 33].
count	The number of encoded text strings.
encoded_text	An array of encoded text strings [§2.13.1, page 37].

Returns

None.

Description

Invoke **PEXOCCEncodedText2D** to create a two-dimensional text primitive output command (**PEXOC-Text2D**).

This primitive is similar to the **PEXText2D** primitive, except that you can specify multiple encoded text strings. 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 implementation uses the default font group to render the entire string. If a character set value is not available in the default font group, then the implementation renders that portion of the string in an implementation-dependent way.

The character set width indicates the width or size of characters in the strings. PEXlib defines the following character set widths: **PEXCSByte**, **PEXCSShort** and **PEXCSLong**. The implementation does not interpret the encoding state; only the application uses it.

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

Errors

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

See Also

For primitives similar to this one, see:**PEXOCCAnnotationText**, **PEXOCCAnnotationText2D, PEXOCCEncodedAnnoText, PEXOCCEncodedAnnoText2D, PEXOCCEncodedText, PEXOCCText,**

PEXOCCText2D.

For PEXlib 5.1 compatibility primitives similar to this one, see: **PEXAnnotationText**, **PEXAnnotationText2D, PEXEncodedAnnoText, PEXEncodedAnnoText2D, PEXEncodedText, PEXEncodedText2D, PEXText, PEXText2D.**

For attributes that affect this primitive, see "Text Attributes" on page 163.

PEXOCCExtendedCellArray - Extended 3D Cell Array Primitive (5.0)

Synopsis

void **PEXOCCExtendedCellArray**(PEXOCC *context*, PEXCoord **point1*, PEXCoord **point2*, PEXCoord **point3*, unsigned int *col_count*, unsigned int *row_count*, PEXPointer *colors*)

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

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
point1	The first cell array definition point [§2.7.1, page 33].
point2	The second cell array definition point [§2.7.1, page 33].
point3	The third cell array definition point [§2.7.1, page 33]
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 (OCC)	The type of color for the cell colors (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
colors	An array of colors specifying the color of each cell [§2.5.11, page 31]. The OCC form of this function allows you to point directly at a list of colors instead of using the PEXArrayOfColor union.

Returns

None.

Description

Invoke **PEXOCCExtendedCellArray** to create a three-dimensional cell array primitive output command (**PEXOCExtendedCellArray**).

This primitive is similar to the **PEXCellArray** primitive, except depending on the *color_type* you specify, this primitive passes the colors as either indexed color or direct color values.

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

For primitives similar to this one, see: **PEXOCCCellArray**, **PEXOCCCellArray2D.**

For PEXlib 5.1 compatibility primitives similar to this one see: **PEXCellArray, PEXCellArray2D**,

PEXExtendedCellArray

For attributes that affect this primitive, see "Surfaces - General Attributes" on page 164.

PEXOCCFillArea - Fill Area Primitive (5.0)

Synopsis

void **PEXOCCFillArea**(`PEXOCC` *context*, `PEXPointer` *facet_data*, unsigned int *count*, `PEXPointer` *vertices*)

Arguments

context — An OC context returned by a successful **PEXCreateOCC** call.

display (OCC) — A pointer to a display structure returned by a successful **XOpenDisplay** call.

resource_id (OCC) — The resource identifier of the renderer or structure.

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

shape_hint (OCC) — The shape of the fill area (**PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex** or **PEXShapeUnknown**).

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

facet_attributes (OCC)
A mask indicating the facet attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.0), **PEXGANormal**(5.0), or **PEXGAFloatData**(5.2)).

surface_vertex_attributes (OCC)
A mask indicating the vertex attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.0), **PEXGANormal**(5.0), **PEXGAEdges**(5.0), **PEXGA2D**(5.2), **PEXGAFloatData**(5.2)).

color_type (OCC) — The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).

facet_fp_data_count (OCC)
The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.

facet_data — A pointer to facet data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

count — The number of vertices.

vertices — A pointer to the vertex data defining the fill area [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

data_model (OCC) — Specifies the data model used to represent the geometric data.

data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCFillArea** to create a fill area primitive output command. This function encompasses all the functionality contained in the **PEXFillArea**, **PEXFillArea2D**, and **PEXFillAreaWithData** functions. Depending on the attributes you supply, this function generates a **PEXOCFillArea**, **PEXOCFillArea2D**, or **PEXOCFillAreaSetWithDataFP** output command. If a **PEXOCFillAreaSetWithDataFP** OC is needed and the *display* is a connection to a PEX 5.1 server or the Alternate Protocol Level is 5.1, this function generates a **PEXOCFillAreaSetWithData** output command instead of a **PEXOCFillAreaSetWithDataFP** output command.

A fill area is a list of vertices joined together to form a planar surface. The implementation does not require fill areas to be strictly 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 implementation implicitly connects the last vertex to the first.

During the rendering process, the implementation transforms the fill area vertices to positions in device coordinates.

Reflectance calculations affect the surface colors. These calculations use the light state, interior style, and reflection model attributes. Depth-cueing computations affect surface colors and map the surface colors to device colors. The implementation does not display fill areas outside of the currently defined clipping volume. The implementation clips fill areas crossing the clipping volume, and displays only the portions inside the clipping volume.

Defining a fill area with fewer than three vertices creates a degenerate area. The implementation stores the fill area in a structure, but when the area is rendered, there is no visual effect. The implementation ignores the primitive.

A fill area may cross over itself to create a complex shape. The implementation uses the even-odd rule to determine the area that lies in the interior of the fill area. Specifying the *shape_hint* argument improves performance for certain shapes. The implementation renders fill areas that are of higher complexity than indicated by *shape_hint* in an implementation-dependent way. Consequently, your applications should pass **PEXShapeUnknown** as the shape unless you are certain the fill area's shape is one of the other three.

Specify the *ignore_edges* argument as a boolean value indicating whether surface edges are rendered. This argument value applies to each of the fill areas in the set. If **True**, the implementation draws no edges for the fill area set, even if edge data is supplied and/or the pipeline state's surface edge flag is on. If **False**, the implementation draws edges only if the pipeline state's surface edge flag is on. If you set **PEXGAEdges** in the vertex attributes, you supply a surface edge flag for each vertex by setting the edge field of the vertex data to either **PEXOn** or **PEXOff**. For the i^{th} vertex, the edge flag indicates whether to draw the edge between the i_{th} and $i+1_{th}$ vertex. If you do not supply edge data by not setting the **PEXGAEdges** vertex attribute, then the implementation draws all edges of the fill area sets.

Depending on the setting of the surface edge source flag values, this primitive obtains the attributes: surface edges, surface edge color, surface edge type, and surface edge width either 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 attribute source flag values, this primitive obtains the attributes: surface color, interior style, interior style index, surface interpolation method, and reflection model either directly from the current surface attributes values or from the interior bundle lookup table entry specified by the current interior bundle index attribute.

If a surface is front-facing with respect to the point of view and the current culling mode enables front-faces to be rendered, then the implementation uses the surface color and reflection attributes when rendering the surface. If the surface is back-facing, and the current facet distinguish mode is **True**, and the current culling mode enables back-facing faces to be rendered, then the implementation uses the corresponding back-facing attributes instead.

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

Specify *facet_attributes* to 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. If specified, facet colors override the current surface color.

Specify *surface_vertex_attributes* to indicate the content of each fill area vertex. In addition to the coordinate (*x,y,z*), your application may specify a color, a normal, 2D coords, or optional floating-point data for each vertex. Use the constants **PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGA2D**, and **PEXGAFloatData** to construct a mask indicating the data provided. If specified, vertex colors 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 affect how the implementation uses additional data in rendering the surface.

Normals are assumed to be unit length vectors. If you specify a normal that is not unit length, then the result is implementation-dependent.

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

For primitives similar to this one, see: **PEXOCCFillAreaSet**, **PEXOCCIndexedFillAreaSets.**

For PEXlib 5.1 compatibility primitives similar to this one see: **PEXFillArea**, **PEXFillArea2D**, **PEXFillAreaWithData**.

For attributes that affect this primitive, see, “Data Mapping Attributes” on page 159, “Edge Attributes” on page 160, “Surfaces - General Attributes” on page 164, “Surfaces - Styled and Shaded Attributes” on page 167.

PEXOCCFillAreaSet - Fill Area Set Primitive (5.0)

Synopsis

void **PEXOCCFillAreaSet**(`PEXOCC` *context*, unsigned int *count*, `PEXPointer` *facet_data*,
`PEXListOfVertexData` **vertex_lists*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
shape_hint (OCC)	The shape which describes all of the contours (**PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex** or **PEXShapeUnknown**).
ignore_edges (OCC)	A flag that determines if surface edges are rendered (**True** or **False**).
contour_hint (OCC)	A flag that indicates whether contours are disjoint or overlapping (**PEXContourDisjoint**, **PEXContourNested**, **PEXContourIntersecting** or **PEXContourUnknown**).
facet_attributes (OCC)	A mask indicating the facet attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.0), **PEXGANormal**(5.0) or **PEXGAFloatData**(5.2)).
surface_vertex_attributes (OCC)	A mask indicating the vertex attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.0), **PEXGANormal**(5.0), **PEXGAEdges**(5.0), **PEXGAFloatData**(5.2) or **PEXGA2D**(5.2)).
color_type (OCC)	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
facet_fp_data_count (OCC)	The number of optional floating-point numbers for each facet.
surface_vertex_fp_data_count (OCC)	The number of optional floating-point numbers for each vertex.
count	The number of fill areas in the set.
facet_data	A pointer to facet data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.
vertex_lists	A pointer to the list of vertex arrays defining each contour of the fill area set [§2.21.5, page 50]. See "Facet / Vertex Data Formats" on page 137.
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCFillAreaSet** to create a fill area set primitive output command. This function encompasses all the functionality contained in the **PEXFillAreaSet**, **PEXFillAreaSet2D**, and **PEXFillAreaSetWithData** functions. Depending on the attributes you supply, this function generates a **PEXOCFillAreaSet**, **PEXOCFillAreaSet2D**, or **PEXOCFillAreaSetWithDataFP** output command. If a **PEXOCFillAreaSetWithDataFP**

OC is needed and the *display* is a connection to a PEX 5.1 server or the Alternate Protocol Level is 5.1, this function generates a **PEXOCFillAreaSetWithData** output command instead of a **PEXOCFillAreaSetWithDataFP** output command.

This primitive is similar to **PEXOCCFillArea** but enables you to create areas with "islands" or "holes".

Defining any fill area in the set with fewer than three vertices or without any contours, creates a degenerate area. The implementation stores the primitive in a structure, but when the area is rendered, there is no visual effect. The implementation ignores the primitive.

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.

Specify the *contour_hint* argument to provide information about the relationships between contours in the fill area set:

> **PEXContourDisjoint -** indicates that all contours are spatially disjoint. There is no overlapping or intersection between any contours in the fill area set.
>
> **PEXContourNested** - indicates that contours are either disjoint or wholly contained within another contour. No contour has edges that intersect or are coincident with edges of any other contour.
>
> **PEXContourIntersecting** - indicates that separated contours may have edges that are coincident or overlap.
>
> **PEXContourUnknown** - indicates that nothing is known about the interrelationships between contours.

The implementation renders fill area sets, with contours that have higher complexity inter-relationships than that indicated by the contour hint, in an implementation-dependent way.

Specify the *ignore_edges* argument as a boolean value indicating whether surface edges are rendered. This argument value applies to each of the fill areas in the set. If **True**, the implementation draws no edges for the fill area set, even if edge data is supplied and/or the pipeline state's surface edge flag is on. If **False**, the implementation draws edges only if the pipeline state's surface edge flag is on. If you set **PEXGAEdges** in the vertex attributes, you supply a surface edge flag for each vertex by setting the edge field of the vertex data to either **PEXOn** or **PEXOff**. For the i^{th} vertex, the edge flag indicates whether to draw the edge between the i^{th} and $i{+}1^{th}$ vertex. If you do not supply edge data by not setting the **PEXGAEdges** vertex attribute, then the implementation draws all edges of the fill area sets.

The facet attributes indicate the content of the facet data. Use the constants **PEXGANone**, **PEXGAColor**, **PEXGANormal** and **PEXGAFloatData** to construct a mask indicating the data provided. Use the PEXlib-supplied packed format data structures to map the facet data if using the packed data interface to maintain the order of color, normal, float_data.

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 surface vertex attributes indicate the content of each fill area vertex. Use the constants **PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGAFloatData** and **PEXGA2D** to construct a mask indicating the data provided. Use the PEXlib-supplied packed format data structures to map the vertex data if using the packed data interface to maintain the order of color, normal, edges, float_data.

If specified, vertex colors 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 affect how the implementation uses additional data in rendering the surface.

Normals are assumed to be unit length vectors. If you specify a normal that is not unit length, then the result is implementation-dependent.

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

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

For primitives similar to this one, see: **PEXOCCFillArea, PEXOCCIndexedFillAreaSets.**

For PEXlib 5.1 compatibility primitives similar to this one see: **PEXFillAreaSet**, **PEXFillAreaSet2D**, **PEXFillAreaSetWithData**.

For attributes that affect this primitive, see, "Data Mapping Attributes" on page 159, "Edge Attributes" on page 160, "Surfaces - General Attributes" on page 164, "Surfaces - Styled and Shaded Attributes" on page 167.

PEXOCCGDP - Generalized Drawing Primitive (5.0)

Synopsis

void **PEXOCCGDP**(`PEXOCC` *context*, long *gdp_id*, unsigned int *count*, `PEXPointer` *occ_points*, unsigned long *length*, char **data*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
gdp_id	The identifier of the GDP.
count	The number of vertices.
occ_points	A pointer to the vertex data used by the GDP [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.
points	The vertices used by the GDP [§2.7.1, page 33].
length	The length, in bytes, of the data.
data	Additional data used by the GDP.

Returns

None.

Description

Invoke **PEXOCCGDP** to create a generalized drawing primitive output command (**PEXOCGDP**).

Individual PEX implementations should provide you with a complete interface and description for each GDP identifier. PEXlib does not define any standard PEX GDPs. If your implementation does not support the specified GDP identifier, then it ignores the output command generated by this function.

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

PEXOCCIndexedFillAreaSets - Indexed Fill Area Sets (SOFAS) Primitive(5.0)

Synopsis

void **PEXOCCIndexedFillAreaSets**(`PEXOCC` *context*, unsigned int *set_count*, `PEXPointer` *occ_facet_data*, unsigned int *vertex_count*, `PEXPointer` *occ_vertices*, unsigned int *index_count*, `PEXSwitch` **edge_flags*, `PEXConnectivityData` **connectivity*)

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

context — An OC context returned by a successful **PEXCreateOCC** call.

display (OCC) — A pointer to a display structure returned by a successful **XOpenDisplay** call.

resource_id (OCC) — The resource identifier of the renderer or structure.

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

shape_hint (OCC) — The shape which describes all of the contours (**PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex** or **PEXShapeUnknown**).

facet_attributes (OCC)
A mask indicating the facet attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.0), **PEXGANormal**(5.0) or **PEXGAFloatData**(5.2)).

surface_vertex_attributes (OCC)
A mask indicating the vertex attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.0), **PEXGANormal**(5.0), **PEXGAFloatData**(5.2) or **PEXGA2D**(5.2)).

edge_attributes (OCC)
A mask indicating the edge attributes provided (**PEXGANone** or **PEXGAEdges**).

contour_hint (OCC) — A flag indicating whether contours are disjoint or overlapping (**PEXContourDisjoint**, **PEXContourNested**, **PEXContourIntersecting** or **PEXContourUnknown**).

contours_all_one (OCC)
True if each fill area set contains only one contour; **False** otherwise.

color_type (OCC) — The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).

facet_fp_data_count (OCC)
The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.

set_count — The number of fill area sets.

occ_facet_data — A pointer to the facet data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

facet_data — An array of facet data [§2.12.11, page 36].

vertex_count — The number of vertices.

occ_vertices — A pointer to the vertex data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

vertices — An array of vertices [§2.32.87, page 79].

index_count — The number of vertex connectivity indices (also number of edge flags, if edges are specified).

edge_flags	An array of edge flags [§2.3.20, page 29].
connectivity	A pointer to the list of contour connectivity data [§2.6.2, page 33].
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCIndexedFillAreaSets** to create a set of fill area sets primitive output command (**PEXOCSetOfFillAreaSetsFP**). If the *display* is a connection to a PEX 5.1 server or the Alternate Protocol Level is 5.1, this function generates a **PEXOCSetOfFillAreaSets** output command instead of a **PEXOCSetOfFillAreaSetsFP** output command.

A fill area may cross over itself to create a complex shape. PEXlib uses the even-odd rule to determine the area that lies in the interior of the fill area. Specifying the *shape_hint* argument improves performance for certain shapes. The implementation renders fill areas that are of higher complexity than indicated by *shape_hint* in an implementation-dependent way. Consequently, applications should pass **PEXShapeUnknown** as the shape unless you are certain the fill area's shape is one of the other three.

A fill area set with more than one contour is always allowed to have contours that intersect. Rendering optimization could occur only 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**.

Specify the *contour_hint* argument to provide information about the relationships between contours in the fill area set:

PEXContourDisjoint - indicates that all contours are spatially disjoint. There is no overlapping or intersection between any contours in the fill area set.

PEXContourNested - indicates that contours are either disjoint or wholly contained within another contour. No contour has edges that intersect or are coincident with edges of any other contour.

PEXContourIntersecting - indicates that separated contours may have edges that are coincident or overlap.

PEXContourUnknown - indicates that nothing is known about the interrelationships between contours.

The implementation renders fill area sets with contours that have higher complexity inter-relationships than that indicated by the *contour hint* in an implementation-dependent way.

The facet attributes indicate the content of *facet_data*. Use the constants **PEXGANone**, **PEXGAColor**, **PEXGANormal** and **PEXGAFloatData** 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. You must provide data in the *facet_data* argument for each facet (fill area set) of the set of fill area sets, if you've indicated facet data with the facet attributes. Be aware that even if you define a fill area set has zero contours defined, you still need to provide a facet data entry that corresponds to that fill area set.

The vertex attributes indicate the content of each fill area vertex. Use the constants **PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAFloatData** and **PEXGA2D** to construct a mask indicating the data provided. If specified, vertex colors 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 affect how the implementation uses the additional data when rendering the surface.

Color values passed must be of the specified color type. Normals are assumed to be unit length vectors. If you specify a normal that is not unit length, then the result is implementation-dependent.

The edge attributes indicate the content of *edge_flags*. Use **PEXGAEdges** to indicate that the application provides edge flags or use **PEXGANone** if the application does not provide edge flags. The implementation uses these edge flags to draw edges if the pipeline state’s surface edge flag is on. The edge flags, if present, are set to **PEXOn** or **PEXOff** and indicate which edges should be rendered. The edge flag list corresponds to the order in which vertices are specified in *connectivity*, even if the vertices form a degenerate contour which is not drawn. The first edge flag in *edge_flags* corresponds to the edge between the first and second vertices of the first contour of the first fill area set, and so on. The edge flag corresponding with the last vertex of each contour indicates whether a vertex is drawn from the last vertex to the first vertex of the contour. If you do not provide edge data and the pipeline state’s surface edge flag is on, then the implementation draws all the edges. The implementation always renders surface edges with the surface edge color. Facet or vertex colors do not affect surface edges.

The connectivity of the primitive is defined by *connectivity.* 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. It is implementation dependent whether a true value of this flag allows some of the contour counts to be zero.

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

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

For primitives similar to this one, see: **PEXOCCFillArea, PEXOCCFillAreaSet.**

For PEXlib 5.1 compatibility primitives similar to this one see: **PEXSetOfFillAreaSets**.

For attributes that affect this primitive, see, “Data Mapping Attributes” on page 159, “Edge Attributes” on page 160, “Surfaces - General Attributes” on page 164, “Surfaces - Styled and Shaded Attributes” on page 167.

PEXOCCIndexedTriangles - Indexed Triangle Primitive (5.2)

Synopsis

void **PEXOCCIndexedTriangles**(PEXOCC *context*, PEXPointer *facet_data*, unsigned int *num_verts*, PEXPointer *vertices*, unsigned int *index_count*, PEXSwitch **per_edge_data*, unsigned short **connectivity*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
facet_attributes (OCC)	A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).
surface_vertex_attributes (OCC)	A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAFloatData** or **PEXGA2D**).
edge_attributes (OCC)	A mask indicating the edge attributes provided (**PEXGANone**, **PEXGAEdges**).
color_type (OCC)	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
facet_fp_data_count (OCC)	Number of floats per facet provided.
surface_vertex_fp_data_count (OCC)	Number of floats per vertex provided.
facet_data	A pointer to the facet data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.
num_verts	Number of vertices.
vertices	A pointer to the vertex data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.
index_count	The number of vertex connectivity indices (also number of edge flags, if edges are specified).
per_edge_data	Optional edge data [§2.3.20, page 29].
connectivity	A pointer to the list of triangle connectivity data.
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCIndexedTriangles** to create a triangle set primitive output command (**PEXOCIndexedTriangles**).

This primitive independently renders each triangle in a set of triangles. A triangle set is a collection of triangles defined by reference into a modeling coordinates vertex list. The *vertices* are defined in a list of length *num_-verts* such that the first vertex is referenced as vertex 0. The vertices that make up each triangle are defined by the *connectivity* array which consists of *index_count* indices into the *vertices*. Each three successive connectivity indices reference the three vertices that make up a triangle. The implementation draws *index_count*/3 triangles If index_count is not a multiple of three, the extra indices and vertices are ignored. If you indicate that facet data is present with the facet attributes, you must supply facet data for each facet (triangle), which is *index_count*/3 facet data items.

If you define facet and/or vertex colors, then they are of the type specified by *color_type*. Otherwise the implementation ignores *color_type*.

Facet and vertex normals are assumed to be unit length vectors. If you specify a normal that is not unit length, then the result is implementation-dependent.

If you set the **PEXGAFloatData** bit of the facet attributes and/or vertex attributes, then you must supply extended floating-point data with each facet and/or vertex, respectively. If the **PEXGAFloatData** facet attribute bit is set, then *facet_fp_data_count* indicates the number of floating-point values for each facet. If you set the **PEXGAFloatData** vertex attribute bit, then *surface_vertex_fp_data_count* indicates the number of floating-point values for each vertex. If you set only one floating-point data attribute bit, then the implementation ignores the count corresponding to the other floating-point data attribute bit. If neither floating-point data attribute bit is set, then the implementation ignores both *facet_fp_data_count* and *surface_vertex_fp_data_-count*.

If you set the **PEXGA2D** vertex attribute bit, then the vertex coordinates are of type COORD_2D instead of type COORD_3D.

Edge flags are present in *per_edge_data* if the **PEXGAEdges** bit is set in *edge_attributes*. This edge data is used if the pipeline state's *surface edge flag* is **PEXOn**. *per_edge_data* is a list of *index_count* entries, each of which contains a switch (set to either **PEXOn** or **PEXOff**) indicating whether the corresponding edge is drawn. Each edge flag specifies whether the edge between vertices is to be rendered. The list corresponds to the order in which vertices and triangles are specified in *connectivity*, with the first edge flag in *per_edge_data* corresponding to the edge between the first and second vertices of the first triangle, the second edge flag corresponding to the edge between the second and third vertices, the third edge flag corresponding to the edge between the third and first vertices of the triangle, and so on for each triangle. If the *edges* bit is not set, the *per_edge_data* list is empty. If no edge data is supplied, all the edges are drawn if the pipeline state's *surface edge flag* is **PEXOn**.

Clipping is similar to Fill Area 3D. See "Surfaces - General Attributes" on page 164..

The basic surface, styled and shaded surface, edge, and data mapping attribute sets are applied to this primitive.

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

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

For primitives similar to this one, see: **PEXOCCPolyTriangle, PEXOCCTriangleFan, PEXOCCTriangles, PEXOCCTriangleStrip.**

There are no PEXlib 5.1 compatibility primitives similar to this one.

For attributes that affect this primitive, see, “Data Mapping Attributes” on page 159, “Edge Attributes” on page 160, “Surfaces - General Attributes” on page 164, “Surfaces - Styled and Shaded Attributes” on page 167.

PEXOCCMarkers - 3D Markers Primitive (5.0)

Synopsis

void **PEXOCCMarkers**(`PEXOCC` *context*, unsigned int *count*, `PEXPointer` *occ_points*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
marker_vertex_attributes(OCC)	A mask indicating the vertex attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.2)**,** **PEXGA2D**(5.0), or **PEXGAFloatData**(5.2)).
marker_vertex_fp_data_count(OCC)	The number of optional floating-point numbers for each vertex.
color_type (OCC)	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
count	The number of points.
occ_points	A pointer to vertex data specifying marker locations [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.
points	An array of points specifying marker locations [§2.7.1, page 33].
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCMarkers** to create a markers primitive output command. Depending on the attributes you supply, this function generates a **PEXOCMarkers**, **PEXOCMarkers2D**, or **PEXOCMarkerWithData** output command.

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

With the OCC version of this primitive, you can provide color and floating-point data as additional data.

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. Depth-cueing affects the marker's color and maps it to a device color.

If your application positions the marker within the clipping volume but renders it outside of the clipping volume, then the result is implementation-dependent.

Depending on the setting of the marker source flag, this primitive obtains the attributes: marker color, marker type, and marker scale either directly from the current marker attributes or from the marker bundle lookup table entry specified by the current marker bundle index attribute.

Errors

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

See Also

For PEXlib 5.1 compatibility primitives similar to this one see: **PEXMarkers, PEXMarkers2D**.

For attributes that affect this primitive, see, “Marker Attributes” on page 162,

PEXOCCNURBCurve - Non-Uniform Rational B-spline Curve Primitive (5.0)

Synopsis

void **PEXOCCNURBCurve(**`PEXOCC` *context*, unsigned int *shape*, `PEXCurveGeometrySpline` **curve*, `PEXCurveColorSpline` **color_spline*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
shape	Hint indicating the geometrical shape of the curve.
curve	A pointer to a geometry spline data structure [§2.20.8, page 48].
color_spline	A pointer to an optional color spline data structure [§2.20.7, page 47].

Returns

None.

Description

Invoke **PEXOCCNURBCurve** to create a non-uniform B-spline curve primitive output command with or without an optional color spline. Depending on the attributes you supply, this function generates a **PEXOCNURBCurve** or **PEXOCNURBCurveWithColor** output command. Note that the **PEXDataStride** and **PEXDataUnpacked** data models are not supported in this function. You must specify all coordinate, knot and color data in contiguous lists.

Shape is a hint indicating the geometrical shape of the curve. PEXlib defines the following shape values: **PEXNURBSHintUnknown** which indicates that nothing is known about the shape of the curve. The implementation determines the hint itself. In addition, PEXlib defines **PEXNURBSHintLine**, **PEXNURBSHintCircle**, **PEXNURBSHintCircularArc**, **PEXNURBSHintEllipse**, **PEXNURBSHintEllipticalArc**, **PEXNURBSHintParabolicArc**, and **PEXNURBSHintHyperbolicArc**. If the specified *shape* does not match the curve, then the result is implementation-dependent.

Specify the curve itself with the `PEXCurveGeometrySpline` data structure. Specify the curve order as a positive number. Invoke the **PEXGetImpDepConstants** function to determine the largest supported value for curve order. Specify the spline shape by 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* member is **PEXRational**, then your application must specify the control points in homogeneous (4D) modeling coordinates. PEXlib assumes that 4D modeling coordinates (*x,y,z*) have already been multiplied by the homogeneous coordinate (*w*). If *rationality* is **PEXNonRational**, then your application must specify the control points 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 $tmin >= t(order)$, $tmax <= t(k+1-order)$.

If your implementation does not support the specified curve order, then the implementation stores the output primitive in a structure, but when rendering this primitive, there is no visual effect. The implementation ignores this primitive.

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

If you do not supply a color spline, set the argument *color_spline* to NULL. Otherwise, supply a pointer to a `PEXCurveColorSpline` data structure to define the color distribution of the primitive's geometry. The color of any point on the primitive's geometry is determined by evaluating the color spline at the corresponding parametric value. The color spline does not have to have the same order, rationality, or knot sequence as the geometric curve. The parameter range of the color spline must wholly include the bound parameter range of the geometric spline. Each control point's components correspond to a direct color. If the color spline is rational, then the direct color components include a homogeneous coordinate. The color components may be outside the range normally associated with the color type. The number of color spline control points must be at least as large as the color spline's order. The number of knots must be equal to the number of control points plus the order. The knot values sequence must be non-decreasing. If an implementation indicates support for a non-homogeneous color type via the **PEXColorType** enumerated type, then it also supports the homogeneous form of the same color type in the color spline, except for **PEXColorTypeIndexed**.

Depending on the setting of the polyline source flag, this primitive obtains the attributes: line color, line type, line width and curve approximation either directly from the current line attributes or from the line bundle lookup table entry specified by the current line bundle index 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

For PEXlib 5.1 compatibility primitives similar to this one see **PEXNURBCurve**.

For attributes that affect this primitive, see "Curve Attributes" on page 158, "Line Attributes" on page 161, "Lines - Shading Attributes" on page 162

PEXOCCNURBSurface - Non-Uniform Rational B-spline Surface Primitive (5.0)

Synopsis

void **PEXOCCNURBSurface**(`PEXOCC` *context*, unsigned int *shape*, `PEXSurfaceGeometrySpline` **points*, unsigned int *curve_count*, `PEXListOfTrimCurve` **trim_curves*, `PEXSurfaceColorSpline` **color_spline*, unsigned int *num_data_splines*, `PEXSurfaceDataSpline` **data_splines*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
shape	Hint indicating the geometrical shape of the surface.
points	A pointer to a surface geometry spline data structure defining the B-spline surface [§2.20.11, page 49].
curve_count	The number of trimming curves.
trim_curves	A pointer to a list of trimming curve data structures [§2.30.2, page 67].
color_spline	A pointer to a surface color spline data structure [§2.20.9, page 48]
num_data_splines	The number of data splines supplied in the *data_splines* argument.
data_splines	A pointer to a list of surface data spline data structures [§2.20.10, page 48].

Returns

None.

Description

Invoke **PEXOCCNURBSurface** to create a non-uniform B-spline surface primitive output command with optional trimming curves, color spline, and data splines. Depending on the attributes you supply, this function generates a **PEXOCNURBSurface** or **PEXOCNURBSurfaceWithData** output command. Note that the **PEXDataStride** and **PEXDataUnpacked** data models are not supported in this function. You must specify all coordinate, knot, color and float data in contiguous lists.

Shape is a hint indicating the geometrical shape of the surface. PEXlib defines the following shape values: **PEXNURBSHintUnknown** which indicates that nothing is known about the shape of the surface. The implementation determines the hint itself. In addition, PEXlib defines **PEXNURBSHintPlane**, **PEXNURBSHintCone**, **PEXNURBSHintSphere**, **PEXNURBSHintTorus**, **PEXNURBSHintRightCircularCylinder**, **PEXNURBSHintSurfaceOfRevolution**, **PEXNURBSHintTabulatedCylinder**, **PEXNURBSHintRuledSurface**, and **PEXNURBSHintGeneralQuadric**. If the specified *shape* does not match the surface, then the result is implementation-dependent.

Specify the surface with the `PEXSurfaceGeometrySpline` data structure. The surface is generated as a function of the parametric variables *u* and *v*. The *uorder* and *vorder* must be positive integers and indicate the order of the surface in each of the *u* and *v* argument dimensions. Invoke the **PEXGetImpDepConstants** function to determine the largest supported value for surface *uorder* and *vorder*. 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 *uknot* and *vknot* sequences must each form a non-decreasing sequence of numbers. The *col_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 *col_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**, then you must specify the control point list in homogeneous (4D) modeling coordinates. If *rationality* is **PEXNonRational**, then you must specify the control point list in non-homogeneous (3D) modeling coordinates.

In addition to the parametric bounds, you can also specify a list of trimming loops. 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. Close the list explicitly 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, then closure or connectivity is assumed. Trimming loops are defined in the parameter space of the surface and may not go outside the parameter space of the surface.

If the application does not specify any trimming loops, then the implementation renders the rectangular parameter limits of the surface as the edges of the surface, based on the edge flag attribute.

Trimming loops define the region of the surface based on the following two rules: (1) the implementation renders a point that is in the portion of the surface 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 that the implementation trims away should be on the right and the portion the implementation retains should be on the left. In other words, defining a loop in counter-clockwise order causes the implementation to retain the interior of the loop and to trim away the exterior. A clockwise loop causes the implementation to retain the exterior of the loop to and to trim away the interior. If you nest loops, 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. For trimming loops that do not conform to these rules, the result is implementation-dependent.

Each trimming curve has a visibility flag that controls its visibility for the purposes of displaying the surface edges. 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.

Trimming curve specification must abide by the constraints of NURB curve (e.g., the 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.

The color and data attribute splines do not have to have the same orders, rationality, or knot sequences as the geometric surface. The number of attribute spline control points in each argument must be at least as large as the spline's order in that argument. The number of knots in each argument must be equal to the number of control points plus the order in that direction. The parameter ranges of the attribute spline must wholly include the bound parameter ranges of the geometric spline, or, if trimming loops are specified for the geometric surface, then the parameter range must wholly contain the parameter range defined by the trimming loops.

If you do not supply a color spline, set the argument *color_spline* to NULL. Otherwise, supply a pointer to a `PEXCurveColorSpline` data structure to define the color distribution of the primitive's geometry. The color of any point on the primitive's geometry is determined by evaluating the color spline at the corresponding parametric values. Each control point's components correspond to a direct color. If the color spline is rational, then the direct color components include a homogeneous coordinate. The color components may be outside the range normally associated with the color type. If an implementation indicates support for a non-homogeneous color type via the **PEXColorType** enumerated type, then it also supports the homogeneous form of the same color type in the color spline, except for **PEXColorTypeIndexed**.

Specify the number of supplied data splines with the *num_data_splines* argument and supply a pointer to a `PEXSurfaceDataSpline` data structure in the *data_splines* argument to define the distribution over the primitive's geometry of each floating-point value. Each data spline may represent the distribution of any number of floating-point values. A specific floating-point value is determined by evaluating the appropriate data spline at the corresponding parametric values. Each of the control point's components correspond to a specific floating-point value. If a data spline is rational, then the control point additionally includes a homogeneous coordinate. Because there are multiple data splines and each data spline may represent multiple floating-point values, functions referencing the floating-point values must use two indices to indicate a specific floating-point value distribution. If such a functionality only provides one index (as is required for all other primitives with floating-point data), it is assumed this index selects which data spline is to be used. The first floating-point value of that spline is then implicitly selected.

All attributes affecting the representation of fill area sets also affect the representation of the non-uniform B-spline surface primitive. In addition, the implementation uses the surface approximation to determine how to approximate the B-spline surface and uses the parametric surface characteristics to specify the appearance of the surface.

If either of the specified surface orders are not supported, then the implementation stores the output primitive in a structure, but when rendered, there is no visual effect. The implementation ignores the primitive.

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

For PEXlib 5.1 compatibility primitives similar to this one see **PEXNURBSurface**.

For attributes that affect this primitive, see "Edge Attributes" on page 160, "Surfaces - General Attributes" on page 164,"Surfaces - Parametric Attributes" on page 166,"Surfaces - Styled and Shaded Attributes" on page 167

PEXOCCPolyline - Polyline Primitive (5.0)

Synopsis

void **PEXOCCPolyline**(`PEXOCC` *context*, unsigned int *count*, `PEXPointer` *occ_points*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
line_vertex_attributes (OCC)	A mask indicating the vertex attributes provided (**PEXGANone**(5.0)**,** **PEXGAColor**(5.0), **PEXGAEdges**(5.2), **PEXGA2D**(5.0), or **PEXGAFloatData**(5.2)).
line_vertex_fp_data_count (OCC)	The number of optional floating-point numbers for each vertex.
color_type (OCC)	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
count	The number of vertices.
occ_points	A pointer to the vertex data defining the polyline [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.
points	The array of vertices defining the polyline [§2.7.1, page 33].
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCPolyline** to create a primitive output command that defines a single polyline with optional color, move/draw flags, 2D coordinates, or floating-point data. Depending on the attributes you supply, this function generates a **PEXOCPolyline**, **PEXOCPolyline2D,** or **PEXOCPolylineSetWithDataFP** output command. You cannot use the **PEXPolyline** function to specify move/draw flags, 2D coords, color data, or floating-point data. If a **PEXOCPolylineSetWithDataFP** OC is needed and the *display* is a connection to a PEX 5.1 server or the Alternate Protocol Level is 5.1, this function generates a **PEXOCPolylineSetWithData** output command instead of a **PEXOCPolylineSetWithDataFP** output command.

If your application sets the **PEXGAColor** vertex attribute, then the vertex colors are of the type specified by *color_type*. Otherwise the implementation ignores *color_type*.

If colors are passed for each vertex, they are used instead of the line color attribute. Depending on the setting of the polyline source flag, this function obtains polyline interpolation method attributes either directly from the current polyline interpolation attribute or from the line bundle lookup table entry specified by the current line bundle index attribute. This controls how the polylines are shaded.

If your application sets the **PEXGAEdges** vertex attribute, then a move/draw control flag is specified in the first bit of the edge bitmask of each vertex. Using move/draw control flags allows a single polyline to have disjoint components. Therefore, the same disjoint lines can be defined with this function by using multiple polylines without move/draw control flags or by using a single polyline with move/draw control flags. If the move/draw control flag is set to 1 for vertex *i*, a line is drawn between vertex *i*-1 and vertex *i*. If the move/draw control flag is not set, then vertex *i* is the first vertex of the next line. The implementation ignores the move/draw control flag of the first vertex of each polyline. Instead the implementation assumes it is not set.

Note that you are able to specify a disjoint polyline in one of two ways. Use the **PEXOCCPolyline** function with the optional move/draw flags with a single list of vertices, or use the **PEXOCCPolylines** function without move/draw flags and multiple lists of vertices. Note that you are able to draw a list of disjoint polylines by using the **PEXOCCPolylines** function with both move/draw flags and multiple lists of vertices.

If your application sets the **PEXGAFloatData** bit in the vertex attributes bitmask, then you must supply extended floating-point data with each vertex. The *line_vertex_fp_data_count* indicates the number of floating-point values for each vertex. If the **PEXGAFloatData** bit is not set, then *line_vertex_fp_data_count* is ignored.

The vertices, specified in modeling coordinates, are joined together by line segments. The first vertex is joined to the second, the second to the third, and so on. PEXlib does *not* connect the last vertex to the first.

Only depth-cueing affects the polyline colors and maps the polyline colors to device colors. The implementation does not display polylines outside of 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 setting of the polyline source flag, this primitive obtains the attributes: line color, line type, line width, and polyline interpolation method either directly from the current attributes or from the line bundle lookup table entry specified by the current line bundle index attribute.

Defining a polyline with fewer than two vertices creates a degenerate polyline. The implementation stores the polyline in a structure, but when the polyline is rendered, there is no visual effect. The implementation ignores the primitive.

Errors

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

See Also

For primitives similar to this one, see: **PEXOCCPolylines**

For PEXlib 5.1 compatibility primitives similar to this one see **PEXPolyline, PEXPolyline2D, PEXPolylineSetWithData**.

For attributes that affect this primitive, see “Line Attributes” on page 161, “Lines - Shading Attributes” on page 162,

PEXOCCPolylines - Polylines Primitive (5.0)

Synopsis

void **PEXOCCPolylines**(`PEXOCC` *context*, unsigned int *count*, `PEXListOfVertexData` **occ_vertex_lists*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
line_vertex_attributes (OCC)	A mask indicating the vertex attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.0), **PEXGAEdges**(5.2), **PEXGA2D**(5.0) or **PEXGAFloatData**(5.2)).
color_type (OCC)	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
line_vertex_fp_data_count (OCC)	The number of optional floating-point numbers for each vertex.
count	The number of polylines.
occ_vertex_lists	A pointer to a list of vertex arrays defining each polyline in the set [§2.21.5, page 50]. See "Facet / Vertex Data Formats" on page 137.
vertex_lists	A pointer to a list of vertex arrays defining each polyline in the set [§2.32.88, page 81].
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCPolylines** to create a primitive output command that defines multiple polylines with optional color, move/draw flags, 2D coords, or floating-point data (**PEXOCPolylineSetWithDataFP**). You cannot use the **PEXPolylineSetWithData** function to specify move/draw flags, 2D coords, or floating-point data. If the *display* is a connection to a PEX 5.1 server or the Alternate Protocol Level is 5.1, this function generates a **PEXOCPolylineSetWithData** output command instead of a **PEXOCPolylineSetWithDataFP** output command.

This primitive is similar to the **PEXOCCPolyline** primitive, except that it enables your application to specify a number of distinct polylines instead of just one.

Note that you are able to specify a disjoint polyline in one of two ways. Use the **PEXOCCPolyline** function with the optional move/draw flags with a single list of vertices, or use the **PEXOCCPolylines** function without move/draw flags and multiple lists of vertices. Note that you are able to draw a list of disjoint polylines by using the **PEXOCCPolylines** function with both move/draw flags and multiple lists of vertices.

Clipping of this function is similar to Polyline 3D. See "Line Attributes" on page 161..

This primitive uses the line and shaded line attribute sets.

Defining a list of polylines containing a polyline with fewer than two vertices creates a degenerate primitive. The implementation stores the polyline set in a structure when included in a **PEXStoreElements** request, but when the polyline set is rendered there is no visual effect. The implementation ignores the primitive.

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

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

For primitives similar to this one, see: **PEXOCCPolyline**

For PEXlib 5.1 compatibility primitives similar to this one, see **PEXPolylineSetWithData**.

For attributes that affect this primitive, see "Line Attributes" on page 161, "Lines - Shading Attributes" on page 162,

PEXOCCPolyTriangle - Generalized Triangle Primitive (5.2)

Synopsis

void **PEXOCCPolyTriangle**(`PEXOCC` *context*, unsigned int *num_facets*, `PEXPointer` *facet_data*, unsigned int *num_verts*, `PEXPointer` *vertices*)

Arguments

context An OC context returned by a successful **PEXCreateOCC** call.

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

resource_id (OCC) The resource identifier of the renderer or structure.

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

facet_attributes (OCC)
A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).

surface_vertex_attributes (OCC)
A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGAFloatData**, or **PEXGA2D**).

color_type (OCC) The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).

facet_fp_data_count (OCC)
Number of floats per facet provided.

surface_vertex_fp_data_count (OCC)
Number of floats per vertex provided.

num_facets The number of facets.

facet_data A pointer to the facet data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

num_verts The number of vertices.

vertices A pointer to the vertex data defining the polytriangle [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

data_model (OCC) Specifies the data model used to represent the geometric data.

data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCPolyTriangle** to create a primitive output command to independently render each triangle in a set of independent triangles and/or triangle fans and/or triangle strips (**PEXOCTriangleStripFanSetWith-Data**).

A PolyTriangle is defined by a list of vertices in modeling coordinates. The vertices form any combination of triangle subprimitives: triangle strips, triangle fans, and independent triangles.

If you define facet and/or vertex colors, then they are of the type specified by *color_type*. Otherwise the implementation ignores *color_type*.

Facet and vertex normals are assumed to be unit length vectors. If you specify a normal that is not unit length, then the result is implementation-dependent.

If you set the **PEXGAFloatData** bit of the facet attributes and/or surface vertex attributes, then you must supply extended floating-point data with each facet and/or vertex, respectively. If you set the **PEXGAFloatData** facet attribute bit, then *facet_fp_data_count* indicates the number of floating-point values for each facet. If you set the **PEXGAFloatData** vertex attribute bit, then *surface_vertex_fp_data_count* indicates the number of floating-point values for each vertex. If you set only one floating-point data attribute bit, then the implementation ignores the count corresponding to the other floating-point data attribute bit. If neither floating-point data attribute bit is set, then the implementation ignores both *facet_fp_data_count* and *surface_vertex_fp_data_-count*.

If you indicate that facet data is present with the facet attributes, you must supply facet data for each facet (triangle).

If you set the **PEXGA2D** surface vertex attribute, then the vertex coordinates are of type `PEXCoord2D` instead of type `PEXCoord`.

This primitive uses vertex control bits that determine how the vertex is used in the next triangle subprimitive. These bits are stored in the edge data for each vertex, so the edge data must be supplied, by the caller, even if no edge controls are desired. These vertex control bits of the bitmask are required since they are the control bits which define the subprimitives by defining the connectivity of the vertices. Table 5, "Interpretation of Vertex Control Bits," on page 226 shows how the control bits associated with each vertex define how the vertex is used in the definition of a triangle subprimitive.

Table 5: Interpretation of Vertex Control Bits

0x00000008	0x00000004	Description
0	0	The vertex is the first vertex of a triangle subprimitive.
0	1	Drops the oldest vertex of the previous three vertices to form the next triangle in a triangle strip.
1	0	Drops the middle vertex of the three previous vertices to form the next triangle in a triangle fan.
1	1	The vertex defines the second or third vertex in the first triangle of a triangle subprimitive.

The implementation ignores the control bits for the first and second vertices of the PolyTriangle. Instead it is assumed that they are 00 and 11, respectively. Therefore, it is permissible for a PolyTriangle which defines a *single* triangle subprimitive to set the control bits of all the vertices to 01 (for a triangle strip) or 10 (for a triangle fan).

If you set the **PEXGAEdges** bit of *surface_vertex_attributes*, then edge flags are additionally specified in the required vertex edge bitmask. This edge data is used if the pipeline state's *surface edge flag* is **PEXOn**. Table 6, "Interpretation of Edge Flag Bits," on page 227 indicates for each vertex v_i (as defined by the control bits) which edges the bits control. If the *edges* vertex attribute is not set, then all the edges are drawn if the pipeline state's *surface edge flag* is **PEXOn**.

Table 6: Interpretation of Edge Flag Bits

Vertex	Edge Flag Bit	Corresponding Edge
v_0	0x00000001	None
	0x00000002	None
v_1	0x00000001	Edge (v_0, v_1)
	0x00000002	None
v_i[a]	0x00000001	Edge (v_{i-1}, v_i)
	0x00000002	Edge (v_{i-2}, v_i)

a. Where *i* is (2..*n*) inclusive. v_i represents the current vertex, vi-1 the previous vertex, and vi-2 the oldest vertex of the three vertices defining a subprimitive triangle.

Clipping is similar to Fill Area 3D. See "Surfaces - General Attributes" on page 164..

The basic surface, styled and shaded surface, edge, and data mapping attribute sets are applied to this primitive.

If you specify a PolyTriangle that contains one or more subprimitives with fewer than three vertices, then the primitive is stored in a structure when included in a **PEXStoreElements** request. However, the output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request

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

For primitives similar to this one, see: **PEXOCCIndexedTriangles, PEXOCCTriangleFan, PEXOCCTriangles, PEXOCCTriangleStrip.**

There are no PEXlib 5.1 compatibility primitives similar to this one.

For attributes that affect this primitive, see, "Data Mapping Attributes" on page 159, "Edge Attributes" on page 160, "Surfaces - General Attributes" on page 164, "Surfaces - Styled and Shaded Attributes" on page 167.

PEXOCCQuadrilateralMesh - Quadrilateral Mesh Primitive (5.0)

Synopsis

void **PEXOCCQuadrilateralMesh**(`PEXOCC` *context*, `PEXPointer` *occ_facet_data*, unsigned int *col_count*, unsigned int *row_count*, `PEXPointer` *occ_vertices*)

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

context A pointer to an OC Context returned by a successful **PEXCreateOCC** call.

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

resource_id (OCC) The resource identifier of the renderer or structure.

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

shape_hint (OCC) The shape which describes all of the quadrilaterals (**PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex** or **PEXShapeUnknown**).

facet_attributes (OCC)
A mask indicating the facet attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.0), **PEXGANormal**(5.0) or **PEXGAFloatData**(5.2)).

surface_vertex_attributes (OCC)
A mask indicating the vertex attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.0), **PEXGANormal**(5.0), **PEXGAEdges**(5.2), **PEXGAFloatData**(5.2), or **PEXGA2D**(5.2)).

color_type (OCC) The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).

facet_fp_data_count (OCC)
The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.

occ_facet_data An array of facet data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

facet_data A pointer to the facet data [§2.12.11, page 36].

col_count The number of columns in the vertex array.

row_count The number of rows in the vertex array.

occ_vertices A pointer a the vertex data which is interpreted as a two-dimensional (row-major) array defining the quadrilateral mesh [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

vertices A two-dimensional (row-major) array of vertices defining the quadrilateral mesh [§2.32.87, page 79].

data_model (OCC) Specifies the data model used to represent the geometric data.

data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCQuadrilateralMesh** to create a quadrilateral mesh primitive output command (**PEXOCQuadrilateralMeshFP**). If the *display* is a connection to a PEX 5.1 server or the Alternate Protocol Level is 5.1, this function will generate a **PEXOCQuadrilateralMesh** output command instead of a **PEXOCQuadrilateralMeshFP** output command.

The quadrilateral mesh surface is defined by the vertices. The vertex array is accessed in row major order (i.e., the column number varies fastest as vertices are accessed). The (i, j), ((i+1), j), ((i+1), (j+1), and (i, j+1) vertices form to create a single facet. The implementation interconnects adjacent vertices until the entire facet network is processed.

If you set the **PEXGAEdges** bit of the surface vertex attributes, then you specify two surface edge flags with each vertex. This edge data is used if the pipeline state's *surface edge flag* is **PEXOn**. For the (i^{th}, j^{th}) vertex, the first edge flag bit (0x00000001) indicates whether to draw the edge between the ($(i\text{-}1)^{th}$, j^{th}) and (i^{th}, j^{th}) vertices, and the second flag bit (0x00000002) indicates whether to draw the edge between the (i^{th}, $(j\text{-}1)^{th}$) and the (i^{th}, j^{th}) vertex. The first edge flag is ignored for all vertices where i=1 and the second edge flag is ignored for all vertices where j=1. If the *edges* vertex attribute is not set, then all edges are drawn if the pipeline state's *surface edge flag* is **PEXOn**.

If the **PEXGA2D** surface vertex attribute bit is set, then the vertex coordinates are of type COORD_2D instead of type COORD_3D.

If your application does not explicitly provide normals for quadrilaterals, then the normals 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) = (vertex(i+1, j+1) - vertex(i, j)) * (vertex(i, j+1) - vertex(i+1, j))

Normals are assumed to be unit length vectors. If you specify a normal that is not unit length, then 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 **PEXOCCFillArea**.

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

For PEXlib 5.1 compatibility primitives similar to this one see: **PEXQuadrilateralMesh**.

For attributes that affect this primitive, see, “Data Mapping Attributes” on page 159, “Edge Attributes” on page 160, “Surfaces - General Attributes” on page 164, “Surfaces - Styled and Shaded Attributes” on page 167.

PEXOCCSpheres - Sphere Primitive (5.2)

Synopsis

void **PEXOCCSpheres**(`PEXOCC` *context*, double *radius,* unsigned int *count*, `PEXPointer` *spheres*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
color_type (OCC)	The type of color for the spheres (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
surface_vertex_attributes (OCC)	A mask indicating the vertex attributes provided (**PEXGAColor, PEXGARadius, PEXGAFloatData**).
surface_fp_data_count (OCC)	Number of floats provided for each vertex.
radius	The radius of the spheres.
count	The number of spheres.
spheres	A pointer to the sphere vertex data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCSpheres** to create a primitive output command to independently render each sphere in a set of spheres (**PEXOCSphereSetWithData**).

Each sphere is a three-dimensional surface which has points that are all equidistant from its center vertex. Define the center vertex and radius of each circle in modeling coordinates.

For the purposes of surface parameterization, the *u* value is mapped to the full circle contained in the plane of constant Y that intersects the center of the sphere, with the radius of the circle the same as that of the sphere. The point at which *u* = 0.0 is the point on this circle at which X equals *sphere_center.x* + *radius*. The value of *u* increases as the circle is swept in a counter-clockwise direction when viewed from a point above (larger Y value) than the plane of the circle. *u* reaches the value of 1.0 as the starting point of the sweep (at *u* = 0.0) is reached.

For purposes of surface parameterization, the *v* value is mapped to the full circle contained in the plane of constant X that intersects the center of the sphere, with the radius of the circle the same as that of the sphere. The point at which *v* = 0.0 is the point on this circle where Z equals *sphere_center.z* + *radius*. The value of *v* increases as the circle is swept in a counter-clockwise direction when viewed from a point to the right (larger X value) than the plane of the circle. *v* reaches the value of 1.0 as the starting point of the sweep (at *v* = 0.0) is reached.

Surface_vertex_attributes indicates the optional data specified for the center vertex of each sphere. The supported components of the bitmask are **PEXGAColor, PEXGARadius** and **PEXGAFloatData**.

> If the **PEXGAColor** bit is set, then a color is specified with each center vertex. The colors are of the type specified by *color_type*. Otherwise the implementation ignores *color_type*.
>
> If the **PEXGARadius** bit is set, then the radius for each sphere is defined in *spheres* and the *radius* argument is ignored. Otherwise the *radius* argument defines the common radius applied to all spheres in the set. If you specify a radius that is less than zero, then the result is implementation-dependent.
>
> If the **PEXGAFloatData** bit is set, then extended data is specified with each center vertex. The *surface_vertex_fp_data_count* indicates the number of floating-point values for each vertex. If the **PEXGAFloatData** bit is not set, then the implementation ignores *surface_vertex_fp_data_count*.

Clipping is similar to Fill Area 3D. See “Surfaces - General Attributes” on page 164..

PEXlib does not supply a set of type definitions for spheres due to the large number of combinations of vertex data possible when considering both the required and optional vertex data. Therefore, you may find it convenient to design your own data structure to pass in the *spheres* parameter when using the **PEXDataPacked** data model. You need to design the data structure with the following ordering rules in mind:

The required order is:

```
      PEXCoord          center
      PEXColor*         one of the PEXlib color types, if provided
      float             radius
      float[n]          floating point data, if provided
```

As an example, if you wish to supply data for spheres using coordinates, RGB floating-point colors, and a radius, you would create a structure like this:

```
typedef struct {
      PEXCoord          center;
      PEXColorRGB       rgb;
      float             radius;
} MY_PEXRGBSphereVertex;
```

Remember to set the correct surface vertex attributes in the OCC. In this case you would make sure that the **PEXGAColor** flag is on and the **PEXGARadius** flag is on. You set the *color_type* field in the OCC to **PEXColorTypeRGB**.

If you are using the **PEXDataStride** data model, then you set the stride and offset members in the OC Context to reflect the way the data is stored in your structured array. Pass a pointer to the start of the structured array in the *spheres* argument of the function call. Be sure that there is a valid offset for each vertex attribute that is present.

If you are using the **PEXDataUnpacked** data model, then you would set size members in the OCC to the appropriate values and point to each of the data lists with the “PEXUnpackedVertexData Data Structure” on page 51. Again, be careful to set a size and pointer for every attribute you indicate as present in the vertex attributes.

Errors

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

See Also

For primitives similar to this one, see: **PEXOCCCones.**

There are no PEXlib 5.1 compatibility primitives similar to this one.

For attributes that affect this primitive, see "Curve Attributes" on page 158, "Data Mapping Attributes" on page 159, "Surfaces - General Attributes" on page 164, "Surfaces - Styled and Shaded Attributes" on page 167.

PEXOCCText - 3D Text Primitive (5.0)

Synopsis

void **PEXOCCText**(`PEXOCC` *context*, `PEXCoord` **origin*, `PEXVector` **vector1*, `PEXVector` **vector2*, int *length*, char **string*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
origin	The origin of the text string [§2.7.1, page 33].
vector1	A vector defining the positive *x*-direction of the text local coordinate system [§2.31.1, page 67].
vector2	A vector defining the positive *y*-direction of the text local coordinate system [§2.31.1, page 67].
length	The number of bytes in the text string.
string	A pointer to the text string.

Returns

None.

Description

Invoke **PEXOCCText** to create a text primitive output command (**PEXOCText**).

This primitive only uses the first character set in the text font.

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, then PEXlib uses the vector values <1,0,0> and <0,1,0>.

During rendering, the string's position is transformed to a position in device coordinates. Only depth-cueing affects the string's color and maps it to a device color. Clipping of the text string depends on the current text precision attribute:

> **PEXStringPrecision** - clips text in an implementation-dependent way.
>
> **PEXCharPrecision** - clips text done on at least a character-by-character basis.
>
> **PEXStrokePrecision** - performs clipping at the clipping boundaries for each character.

Depending on the setting of the text attribute source flag values, this primitive obtains the attributes: text color, text precision, character expansion, character spacing, and text font either directly from the current text attribute values or from the entry in the text bundle table entry specified by the current text bundle index attribute. The primitive also uses the current character height, text path, text alignment attributes and character up vector to render the text string. The directions specified by the character up vector and text path are relative to the text local coordinate system.

Errors

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

See Also

For primitives similar to this one, see:**PEXOCCAnnotationText**, **PEXOCCAnnotationText2D, PEXOCCEncodedAnnoText, PEXOCCEncodedAnnoText2D, PEXOCCEncodedText, PEXOCCEncodedText2D, PEXOCCText2D.**

For PEXlib 5.1 compatibility primitives similar to this one, see: **PEXAnnotationText**, **PEXAnnotationText2D, PEXEncodedAnnoText, PEXEncodedAnnoText2D, PEXEncodedText, PEXEncodedText2D, PEXText, PEXText2D.**

For attributes that affect this primitive, see “Text Attributes” on page 163.

PEXOCCText2D - 2D Text Primitive (5.0)

Synopsis

void **PEXOCCText2D**(PEXOCC *context*, `PEXCoord2D` **origin*, int *length*, char **string*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display(OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type(OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
origin	The origin of the text string [§2.7.2, page 33].
length	The number of bytes in the text string.
string	A pointer to the text string.

Returns

None.

Description

Invoke **PEXOCCText2D** to create a two-dimensional text primitive output command (**PEXOCText2D**).

This primitive functions like the **PEXText** primitive, except that you need only specify the x- and y-components to define the origin position. The z-component is always assumed to be zero. This primitive is two-dimensional only in that the z-component is implied. PEXlib still carries out the geometry transformations in three dimensions.

Errors

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

See Also

For primitives similar to this one, see:**PEXOCCAnnotationText**, **PEXOCCAnnotationText2D, PEXOCCEncodedAnnoText, PEXOCCEncodedAnnoText2D, PEXOCCEncodedText, PEXOCCEncodedText2D, PEXOCCText.**

For PEXlib 5.1 compatibility primitives similar to this one, see: **PEXAnnotationText**, **PEXAnnotationText2D, PEXEncodedAnnoText, PEXEncodedAnnoText2D, PEXEncodedText, PEXEncodedText2D, PEXText, PEXText2D.**

For attributes that affect this primitive, see "Text Attributes" on page 163.

PEXOCCTriangleFan - Triangle Fan Primitive (5.2)

Synopsis

void **PEXOCCTriangleFan**(`PEXOCC` *context*, `PEXPointer` *facet_data*, unsigned int *count*, `PEXPointer` *vertices*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
facet_attributes (OCC)	A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).
surface_vertex_attributes (OCC)	A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGAFloatData** or **PEXGA2D**).
color_type (OCC)	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
facet_fp_data_count (OCC)	Number of floats per facet provided.
surface_vertex_fp_data_count (OCC)	Number of floats per vertex provided.
facet_data	A pointer to the facet data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.
count	The number of vertices.
vertices	A pointer to the vertex data defining the triangle fan [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCTriangleFan** to create a triangle fan primitive output command (**PEXOCTriangleFan-WithData**).

This primitive independently renders each triangle in a triangle fan. A triangle fan is defined by a list of vertices in modeling coordinates. The first triangle is made up of vertices 1, 2 and 3, the second triangle is made up of vertices 1, 3 and 4, the third triangle is made up of vertices 1, 4 and 5, and so on.

If you define facet and/or vertex colors, then they are of the type specified by *color_type*. Otherwise the implementation ignores *color_type*.

Facet and vertex normals are assumed to be unit length vectors. If you specify a normal that is not unit length, then the result is implementation-dependent.

If you set the **PEXGAFloatData** bit of the facet attributes and/or surface vertex attributes, then you must supply extended floating-point data with each facet and/or vertex, respectively. If you set the **PEXGAFloatData** facet attribute bit, then *facet_fp_data_count* indicates the number of floating-point values for each facet. If you set the **PEXGAFloatData** vertex attribute bit, then *surface_vertex_fp_data_count* indicates the number of floating-point values for each vertex. If you set only one floating-point data attribute bit, then the implementation ignores the count corresponding to the other floating-point data attribute bit. If neither floating-point data attribute bit is set, then the implementation ignores both *facet_fp_data_count* and *surface_vertex_fp_data_-count*.

If you indicate that facet data is present with the facet attributes, you must supply facet data for each facet (triangle).

If you set the **PEXGAEdges** bit of the surface vertex attributes, then two surface edge flags are specified with each vertex. This edge data is used if the pipeline state's *surface edge flag* is **PEXOn**. For the i^{th} vertex, the first edge flag bit (0x00000001) indicates whether to draw the edge between the i^{th} and $(i\text{-}1)^{th}$ vertices, and the second flag bit (0x00000002) indicates whether to draw the edge between the i^{th} and first vertices. The edge flags of the first vertex in *vertices* are ignored; the first edge flag for the second vertex is ignored. If the *edges* vertex attribute is not set, then all edges are drawn if the pipeline state's *surface edge flag* is **PEXOn**.

If the **PEXGA2D** surface vertex attribute bit is set, then the vertex coordinates are of type COORD_2D instead of type COORD_3D.

Clipping is similar to Fill Area 3D. See "Surfaces - General Attributes" on page 164..

The basic surface, styled and shaded surface, edge, and data mapping attribute sets are applied to this primitive.

If you specify a triangle fan containing fewer than three vertices, then the fan is stored in a structure when included in a **PEXStoreElements** request but when rendered, the output command has no visual effect. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

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

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

For primitives similar to this one, see: **PEXOCCIndexedTriangles, PEXOCCPolyTriangle, PEXOCCTriangles, PEXOCCTriangleStrip.**

There are no PEXlib 5.1 compatibility primitives similar to this one.

For attributes that affect this primitive, see, "Data Mapping Attributes" on page 159, "Edge Attributes" on page 160, "Surfaces - General Attributes" on page 164, "Surfaces - Styled and Shaded Attributes" on page 167.

PEXOCCTriangles - Independent Triangles Primitive (5.2)

Synopsis

void **PEXOCCTriangles**(`PEXOCC` *context*, `PEXPointer` *facet_data*, unsigned int *num_vertices*,
`PEXPointer` *vertices*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
facet_attributes (OCC)	A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).
surface_vertex_attributes (OCC)	A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGAFloatData** or **PEXGA2D**).
color_type (OCC)	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
facet_fp_data_count (OCC)	Number of floats per facet provided.
surface_vertex_fp_data_count (OCC)	Number of floats per vertex provided.
facet_data	A pointer to the facet data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.
num_vertices	Number of vertices.
vertices	A pointer to the vertex data defining the triangles [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCTriangles** to create a list of independent triangles primitive output command (**PEXOCTriangles**).

This primitive independently renders each triangle in the list. A triangle list is a collection of triangles defined by the modeling coordinates vertex list. Every three vertices in the vertex list define one independent triangle. *Num_vertices*/3 triangles are rendered. If *num_vertices* is not amultiple of three, the extra vertices are ignored.

If you define facet and/or vertex colors, then they are of the type specified by *color_type*. Otherwise the implementation ignores *color_type*.

Facet and vertex normals are assumed to be unit length vectors. If you specify a normal that is not unit length, then the result is implementation-dependent.

If you set the **PEXGAFloatData** bit of the facet attributes and/or surface vertex attributes, then you must supply extended floating-point data with each facet and/or vertex, respectively. If you set the **PEXGAFloatData** facet attribute bit, then *facet_fp_data_count* indicates the number of floating-point values for each facet. If you set the **PEXGAFloatData** vertex attribute bit, then *surface_vertex_fp_data_count* indicates the number of floating-point values for each vertex. If you set only one floating-point data attribute bit, then the implementation ignores the count corresponding to the other floating-point data attribute bit. If neither floating-point data attribute bit is set, then the implementation ignores both *facet_fp_data_count* and *surface_vertex_fp_data_-count*.

If you indicate that facet data is present with the facet attributes, you must supply facet data for each facet (triangle). You supply *num_vertices*/3 items of facet data.

If you set the **PEXGA2D** vertex attribute bit, then the vertex coordinates are of type COORD_2D instead of type COORD_3D.

If you set the **PEXGAEdges** bit of the vertex attributes, then two surface edge flags are specified with each vertex. This edge data is used if the pipeline state's *surface edge flag* is **PEXOn**. For each triangle in the list, both of the edge flags in the triangle's first vertex are ignored. The first edge flag in the second vertex indicates whether to draw the edge between the first and second vertices. The second edge flag in the second vertex is ignored. The first edge flag in the thrid vertex indicates whether to draw the edge between the second and third vertices. The second edge flag in the third vertex indicates whether to draw the edge between the first and third vertices. If the *edges* vertex attribute is not set, then all the edges are drawn if the pipeline state's *surface edge flag* is **PEXOn**.

Clipping is similar to Fill Area 3D. See "Surfaces - General Attributes" on page 164..

The basic surface, styled and shaded surface, edge, and data mapping attribute sets are applied to this primitive.

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

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

For primitives similar to this one, see: **PEXOCCIndexedTriangles, PEXOCCPolyTriangle, PEXOCCTriangleFan, PEXOCCTriangleStrip.**

There are no PEXlib 5.1 compatibility primitives similar to this one.

For attributes that affect this primitive, see, "Data Mapping Attributes" on page 159, "Edge Attributes" on page 160, "Surfaces - General Attributes" on page 164, "Surfaces - Styled and Shaded Attributes" on page 167.

PEXOCCTriangleStrip - Triangle Strip Primitive (5.0)

Synopsis

void **PEXOCCTriangleStrip**(`PEXOCC` *context*, `PEXPointer` *occ_facet_data*, unsigned int *count*, `PEXPointer` *occ_vertices*)

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

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
facet_attributes (OCC)	A mask indicating the facet attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.0), **PEXGANormal**(5.0) or **PEXGAFloatData**(5.2)).
surface_vertex_attributes (OCC)	A mask indicating the vertex attributes provided (**PEXGANone**(5.0), **PEXGAColor**(5.0), **PEXGANormal**(5.0), **PEXGAEdges**(5.2), **PEXGAFloatData**(5.2), or **PEXGA2D**(5.2)).
color_type (OCC)	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
facet_fp_data_count (OCC)	Number of floats per facet provided.
surface_vertex_fp_data_count (OCC)	Number of floats per vertex provided.
occ_facet_data	A pointer to the facet data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.
facet_data	An array of facet data [§2.12.11, page 36].
count	The number of vertices.
occ_vertices	A pointer to vertex data defining the triangle strip [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.
vertices	An array of vertices defining the triangle strip [§2.32.87, page 79].
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

None.

Description

Invoke **PEXOCCTriangleStrip** to create a triangle strip primitive output command (**PEXOCTriangleStripFP**). If the *display* is a connection to a PEX 5.1 server or the Alternate Protocol Level is 5.1, this function generates a **PEXOCTriangleStrip** output command instead of a **PEXOCTriangleStripFP** output command.

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.

If you define facet and/or vertex colors, then they are of the type specified by *color_type*. Otherwise the implementation ignores *color_type*.

Facet and vertex normals are assumed to be unit length vectors. If you specify a normal that is not unit length, then the result is implementation-dependent.

If you set the **PEXGAFloatData** bit of the facet attributes and/or surface vertex attributes, then you must supply extended floating-point data with each facet and/or vertex, respectively. If you set the **PEXGAFloatData** facet attribute bit, then *facet_fp_data_count* indicates the number of floating-point values for each facet. If you set the **PEXGAFloatData** vertex attribute bit, then *surface_vertex_fp_data_count* indicates the number of floating-point values for each vertex. If you set only one floating-point data attribute bit, then the implementation ignores the count corresponding to the other floating-point data attribute bit. If neither floating-point data attribute bit is set, then the implementation ignores both *facet_fp_data_count* and *surface_vertex_fp_data_count*.

If you indicate that facet data is present with the facet attributes, you must supply facet data for each facet (triangle).

If you set the **PEXGAEdges** bit of the vertex attributes, then two surface edge flags are specified with each vertex. This edge data is used if the pipeline state's *surface edge flag* is **PEXOn**. For the i^{th} vertex, the first edge flag bit of the vertex's edge bitmask indicates whether to draw the edge between the $(i\text{-}1)^{th}$ and i^{th} vertex, and the second bit of the edge bitmask indicates whether to draw the edge between the $(i\text{-}2)^{th}$ and i^{th} vertex. The edge flags of the first vertex in *vertices* are ignored; the second edge flag for the second vertex is ignored. If the *edges* vertex attribute is not set, then all the edges are drawn if the pipeline state's *surface edge flag* is **PEXOn**.

If you set the **PEXGA2D** vertex attribute bit, then the vertex coordinates are of type `PEXCoord2D` instead of type `PEXCoord`.

Clipping is similar to Fill Area 3D. See "Surfaces - General Attributes" on page 164..

The basic surface, styled and shaded surface, edge, and data mapping attribute sets are applied to this primitive.

If you specify a triangle strip containing fewer than three vertices, then the string is stored in a structure when included in a **PEXStoreElements** request but when rendered, the output command has no visual effect. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

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

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

For primitives similar to this one, see: **PEXOCCIndexedTriangles, PEXOCCPolyTriangle, PEXOCCTriangleFan, PEXOCCTriangles.**

For PEXlib 5.1 compatibility primitives similar to this one see: **PEXTriangleStrip**.

For attributes that affect this primitive, see, "Data Mapping Attributes" on page 159, "Edge Attributes" on page 160, "Surfaces - General Attributes" on page 164, "Surfaces - Styled and Shaded Attributes" on page 167.

7.4. PEXlib 5.2 Deprecated Primitive Output Command Functions

PEXFillArea - 3D Fill Area Primitive (5.0)

Note: This function has been superseded by the **PEXOCCFillArea** function, but is maintained here for backwards compatibility with PEXlib 5.1

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** or **PEXShapeUnknown**).
ignore_edges	A flag that determines if surface edges are rendered (**True** or **False**).
count	The number of points.
points	An array of vertices defining the fill area [§2.7.1, page 33].

Returns

None.

Description

Invoke **PEXFillArea** to create a fill area output primitive.

An area is a list of vertices joined together to form a planar surface. PEXlib does not require fill areas to be strictly 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. PEXlib implicitly connects the last vertex to the first.

During the rendering process, the implementation transforms the fill area vertices to positions in device coordinates.

Reflectance calculations affect the surface colors. These calculations use the light state, interior style, and reflection model attributes. Depth-cueing computations affect surface colors and map the surface colors to device colors. The implementation does not display fill areas outside of the currently defined clipping volume. The implementation clips fill areas crossing the clipping volume, and displays only the portions inside the clipping volume.

Defining a fill area with fewer than three vertices creates a degenerate area. The implementation stores the fill area in a in a structure, but when the area is rendered, there is no visual effect. The implementation ignores the primitive.

A fill area may cross over itself to create a complex shape. PEXlib uses the even-odd rule to determine the area that lies in the interior of the fill area. Specifying the *shape_hint* argument improves performance for certain shapes. The implementation renders fill areas that are of higher complexity than indicated by *shape_hint* in an implementation-dependent way. Consequently, your applications should pass **PEXShapeUnknown** as the shape unless you are certain the fill area's shape is one of the other three.

Specify the *ignore_edges* argument as a boolean value indicating whether you want surface edges rendered. If you set the flag to **True**, then the implementation does not render surface edges. If you set the *ignore edges* flag to **False**, then the implementation renders surface edges according to the surface edge attributes if the surface edge flag attribute is **PEXOn**. Depending on the setting of the surface edge source flag values, this primitive obtains the attributes: surface edges, surface edge color, surface edge type, and surface edge width either 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 attribute source flag values, this primitive obtains the attributes: surface color, interior style, interior style index, surface interpolation method, and reflection model either directly from the current surface attributes values or from the interior bundle lookup table entry specified by the current interior bundle index attribute.

If a surface is front-facing with respect to the point of view and the current culling mode enables front-faces to be rendered, then the implementation uses the surface color and reflection attributes when rendering the surface. If the surface is back-facing, then the current distinguish mode is **True**, and the current culling mode enables back-facing surfaces to be rendered, then the implementation uses the corresponding back-facing attributes instead.

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

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

PEXFillArea2D - 2D Fill Area Primitive (5.0)

Note: This function has been superseded by the **PEXOCCFillArea** function, but is maintained here for backwards compatibility with PEXlib 5.1

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** or **PEXShapeUnknown**).
ignore_edges	A flag that determines if surface edges are rendered (**True** or **False**).
count	The number of points.
points	An array of vertices defining the fill area. [§2.7.2, page 33]

Returns

None.

Description

Invoke **PEXFillArea2D** to create a two-dimensional fill area output primitive.

This primitive is similar to the **PEXFillArea** primitive, except that you only need to specify the x- and y-components. The z-component is always assumed to be zero. This primitive is two-dimensional only in that the z-components are implied. The implementation still carries out geometry transformations in three dimensions.

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

PEXFillAreaWithData - 3D Fill Area Primitive With Additional Data (5.0)

Note: This function has been superseded by the **PEXOCCFillArea** function, but is maintained here for backwards compatibility with PEXlib 5.1

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** or **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** or **PEXGANormal**).
vertex_attributes	A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor** or **PEXGANormal**).
color_type	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8** or **PEXColorTypeRGB16**).
facet_data	A pointer to facet data. [§2.12.12, page 37]
count	The number of vertices.
vertices	An array of vertices defining the fill area. [§2.32.87, page 79]

Returns

None.

Description

Invoke **PEXFillAreaWithData** to create a fill area output primitive.

This primitive is similar to the **PEXFillArea** primitive, except that you can specify additional information for the fill area and for each vertex. Color values passed must be of the specified color type.

Specify *facet_attributes* to 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.

Specify *vertex_attributes* to indicate the content of each fill area vertex. In addition to the coordinate (*x,y,z*), your application 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 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 affect how the implementation uses additional data in rendering the surface.

Normals are assumed to be unit length vectors. If you specify a normal that is not unit length, then the result is implementation-dependent.

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

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

PEXFillAreaSet - 3D Set of Fill Areas Primitive (5.0)

Note: This function has been superseded by the **PEXOCCFillAreaSet** function, but is maintained here for backwards compatibility with PEXlib 5.1

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** or **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** or **PEXContourUnknown**).
count	The number of fill areas in the set.
point_lists	A pointer to the list of vertex arrays defining each contour of the fill area set. [§2.7.5, page 33]

Returns

None.

Description

Invoke **PEXFillAreaSet** to create a fill area set output primitive.

This primitive is similar to the **PEXFillArea** primitive, except it enables you to create areas with “islands” or “holes”.

Defining any fill area in the set with fewer than three vertices or without any contours, creates a degenerate area. The implementation stores the primitive in a structure, but when the area is rendered, there is no visual effect. The implementation ignores the primitive.

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.

Specify the *contour_hint* argument to provide information about the relationships between contours in the fill area set:

PEXContourDisjoint - indicates that all contours are spatially disjoint. There is no overlapping or intersection between any contours in the fill area set.

PEXContourNested - indicates that contours are either disjoint or wholly contained within another contour. No contour has edges that intersect or are coincident with edges of any other contour.

PEXContourIntersecting - indicates that separated contours may have edges that are coincident or overlap.

PEXContourUnknown - indicates that nothing is known about the interrelationships between contours.

The implementation renders fill area sets, with contours that have higher complexity inter-relationships than that indicated by the contour hint, in an implementation-dependent way.

Specify the *ignore_edges* argument as a boolean value indicating whether surface edges are rendered. This argument value is applied to each of the fill areas in the set.

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

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

PEXFillAreaSet2D - 2D Set of Fill Areas Primitive (5.0)

Note: This function has been superseded by the **PEXOCCFillAreaSet** function, but is maintained here for backwards compatibility with PEXlib 5.1

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** or **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** or **PEXContourUnknown**).
count	The number of fill areas in the set.
point_lists	A pointer to the list of vertex arrays defining each contour of the fill area set. [§2.7.6, page 33]

Returns

None.

Description

Invoke **PEXFillAreaSet2D** to create a two-dimensional fill area set output primitive.

This primitive is similar to the **PEXFillAreaSet** primitive, except that you only need to specify x- and y-components for the vertices. The z-component is always assumed to be zero. This primitive is two-dimensional only in that the z-components are implied. PEXlib carries out the geometry transformations in three dimensions.

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

PEXFillAreaSetWithData - Set of Fill Areas Primitive With Additional Data (5.0)

Note: This function has been superseded by the **PEXOCCFillAreaSet** function, but is maintained here for backwards compatibility with PEXlib 5.1

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** or **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** or **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** or **PEXColorTypeRGB16**).
count	The number of fill areas in the set.
facet_data	A pointer to facet data. [§2.12.12, page 37]
vertex_lists	A pointer to the list of vertex arrays defining each contour of the fill area set. [§2.32.88, page 81]

Returns

None.

Description

Invoke **PEXFillAreaSetWithData** to create a fill area set output primitive.

This primitive is similar to the **PEXFillAreaSet** primitive, except that you can specify additional information for each fill area and for each vertex. It is similar to **PEXFillAreaWithData** but enables you to create 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. Use the constants **PEXGANone**, **PEXGAColor**, **PEXGANormal** and **PEXGAFloatData** to construct a mask indicating the data provided. Use the PEXlib-supplied packed format data structures to map the facet data if using the packed data interface to maintain the order of color, normal, float_data.

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. Use the constants **PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGAFloatData** and **PEXGA2D** to construct a mask indicating the data provided. Use the PEXlib-supplied packed format data structures to map the vertex data if using the packed data interface to maintain the order of color, normal, edges, float_data.

If specified, vertex colors 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 affect how the implementation uses additional data in rendering the surface. Specify edge controls to indicate which edges the implementation should render when 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. If you specify a normal that is not unit length, then the result is implementation-dependent.

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

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 (5.0)

Note: This function has been superseded by the **PEXOCCGDP** function, but is maintained here for backwards compatibility with PEXlib 5.1

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 vertices.
points	The vertices used by the GDP [§2.7.2, page 33].
length	The length, in bytes, of the data.
data	Additional data used by the GDP.

Returns

None.

Description

Invoke **PEXGDP2D** to create a two-dimensional generalized drawing primitive.

Individual PEX implementations should provide you with a complete interface and description for each GDP identifier. PEXlib does not define any standard PEX GDPs. If your implementation does not support the specified GDP identifier, then it ignores this function.

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

PEXMarkers2D - 2D Markers Primitive (5.0)

Note: This function has been superseded by the **PEXOCCMarkers** output command context function, but is maintained here for backwards compatibility with PEXlib 5.1

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 [§2.7.2, page 33].

Returns

None.

Description

Invoke **PEXMarker2D** to create a two-dimensional markers output primitive.

This primitive is similar to the **PEXMarkers** primitive, except that you only need to specify the x- and y-components. The z-component is always assumed to be zero. This primitive is two-dimensional only in that the z-components are implied. PEXlib carries out the geometry transformations in three dimensions.

Errors

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

See Also

PEXNURBCurve - Non-Uniform Rational B-spline Curve Primitive (5.0)

Note: This function has been superseded by the **PEXOCCNURBCurve** function, but is maintained here for backwards compatibility with PEXlib 5.1

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 [§2.7.4, page 33].
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

Invoke **PEXNURBCurve** to create a non-uniform B-spline curve output primitive.

Specify the curve order as a positive number. Invoke the **PEXGetImpDepConstants** function to determine the largest supported value for curve order. Specify the spline shape by 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* argument is **PEXRational**, then your application must specify the control points in homogeneous (4D) modeling coordinates. PEXlib assumes that 4D modeling coordinates (*x,y,z*) have already been multiplied by the homogeneous coordinate (*w*). If *rationality* is **PEXNonRational**, then your application must specify the control points 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 setting of the polyline source flag, this primitive obtains the attributes: line color, line type, line width and curve approximation either directly from the current line attributes or from the line bundle lookup table entry specified by the current line bundle index attribute.

If your implementation does not support the specified curve order, then the implementation stores the output primitive in a structure, but when rendering this primitive, there is no visual effect. The implementation ignores this primitive.

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

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

PEXNURBSurface - Non-Uniform Rational B-spline Surface Primitive (5.0)

Note: This function has been superseded by the **PEXOCCNURBSurface** function, but is maintained here for backwards compatibility with PEXlib 5.1

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 [§2.7.4, page 33].
curve_count	The number of trimming curves.
trim_curves	A pointer to a list of trimming curves [§2.30.2, page 67].

Returns

None.

Description

Invoke **PEXNURBSurface** to create a non-uniform B-spline surface output primitive.

The surface is generated as a function of the parametric variables *u* and *v*. The *uorder* and *vorder* must be positive integers and indicate the order of the surface in each of the *u* and *v* argument dimensions. Invoke the **PEXGetImpDepConstants** function to determine the largest supported value for surface *uorder* and *vorder*. 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 *uknot* and *vknot* sequences must each form a non-decreasing sequence of numbers. The *col_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 *col_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* argument is **PEXRational**, then you must specify the control point list in homogeneous (4D) modeling coordinates. If *rationality* is **PEXNonRational**, then you must specify the control point list in non-homogeneous (3D) modeling coordinates.

In addition to the parametric bounds, you can also specify a list of trimming loops. 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. Close the list explicitly 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, then closure or connectivity is assumed. Trimming loops are defined in the parameter space of the surface and may not go outside the parameter space of the surface.

If the application does not specify any trimming loops, then the implementation renders the rectangular parameter limits of the surface as the edges of the surface, based on the edge flag attribute.

Trimming loops define the region of the surface based on the following two rules: (1) the implementation renders a point that is in the portion of the surface 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 that the implementation trims away should be on the right and the portion the implementation retains should be on the left. In other words, defining a loop in counter-clockwise order causes the implementation to retain the interior of the loop and to trim away the exterior. A clockwise loop causes the implementation to retain the exterior of the loop to and to trim away the interior. If you nest loops, 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. For trimming loops that do not conform to these rules, result is implementation-dependent.

Each trimming curve has a visibility flag that controls its visibility for the purposes of displaying surface edge. 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 non-uniform B-spline surface primitive. In addition, the implementation uses the surface approximation to determine how to approximate the B-spline surface and uses the parametric surface characteristics to specify the appearance of the surface.

If either of the specified surface orders are not supported, then the implementation stores the output primitive in a structure, but when rendered, there is no visual effect. The implementation ignores the primitive.

Trimming curve specification must abide by the constraints of NURB curve (e.g., the 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.

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

PEXPolyline2D - 2D Polyline Primitive (5.0)

Note: This function has been superseded by the **PEXOCCPolyline** function, but is maintained here for backwards compatibility with PEXlib 5.1

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 vertices.
points	An array of vertices defining the polyline [§2.7.2, page 33].

Returns

None.

Description

Invoke **PEXPolyline2D** to create a two-dimensional polyline output primitive.

This primitive is similar to the **PEXPolyline** primitive, except that you need only to specify x- and y-components. The z-component is always assumed to be zero. This primitive is two-dimensional only in that the z-components are implied. PEXlib still carries out the geometry transformations in three dimensions.

Errors

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

See Also

Chapter 8

Output Attribute Functions

8.1. Output Attribute Functions and Enumerated Types

This section describes all the functions that generate attribute output commands. Functions that use enumerated types for arguments include all the possible values in their descriptions. Use "PEXGetEnumTypeInfo - Get Enumerated Type Information (5.0)" on page 99 to determine which enumerated types are supported by the implementation.

8.2. Attribute Functions and the OC Context

The use of an OC Context with attribute OC functions is much simpler than with primitive OC functions. Only the first three arguments of the PEXlib 5.1 attribute OC functions are in the OC Context, resulting in a fairly straightforward transform from the non-OCC form to the OCC form:

The non-OCC form:

void **PEXfooAttr**(`Display` **display*, `XID` *resource_id*, `PEXOCRequestType` *req_type*, int *arg1*, int *arg2*)

becomes

void **PEXOCCfooAttr**(PEXOCC *context*, int *arg1*, int *arg2*)

For functions that exist in both PEXlib 5.1 and PEXlib 5.2, both forms appear on the page describing the function. For attributes that are new to PEXlib 5.2, only the OCC forms are present.

There are a few exceptions to this rule. Beware of the four transform functions: the OCC form contains a new argument for matrix hints.

PEXOCCAddToNameSet - Add Names to Name Set (5.0)

Synopsis

void **PEXOCCAddToNameSet**(`PEXOCC` *context*, unsigned long *count*, `PEXName` **names*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	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 [§2.3.12, page 28].

Returns

None.

Description

Invoke **PEXOCCAddToNameSet** to create an attribute output command which adds the specified list of *names* to the name set indicated by the renderer's pipeline state.

PEXlib uses included and excluded names to determine whether functions, such as highlighting, invisibility, and picking, should be applied to a primitive based on the pipeline state's name set.

If any *name* is above the maximum supported value for a name, then PEXlib ignores that name. Inquire the maximum supported value for a name by invoking the **PEXGetImpDepConstants** function.

This function is a member of the global attribute set and affects all output primitives.

Errors

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

See Also

PEXCreateNameSet, **PEXGetImpDepConstants, PEXRemoveFromNameSet**

PEXOCCRemoveFromNameSet - Remove Names from Name Set (5.0)

Synopsis

void **PEXOCCRemoveFromNameSet**(`PEXOCC` *context*, unsigned long *count*, `PEXName` **names*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	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 [§2.3.12, page 28].

Returns

None.

Description

Invoke **PEXOCCRemoveFromNameSet** to create an attribute output command which removes the specified list of *names* from the name set indicated by the renderer's pipeline state.

PEXlib uses included and excluded names to determine whether functions, such as highlighting, invisibility, and picking, should be applied to a primitive based on the pipeline state's name set.

If any *name* is above the maximum supported value for a name, then PEXlib ignores that name. Inquire the maximum supported value for a name by invoking the **PEXGetImpDepConstants** function.

This function is a member of the global attribute set and affects all output primitives.

Errors

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

See Also

PEXOCCAddToNameSet, PEXCreateNameSet, **PEXGetImpDepConstants**

PEXOCCRestoreModelClipVolume - Restore Model Clip Volume (5.0)

Synopsis

void **PEXOCCRestoreModelClipVolume**(`PEXOCC` *context*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).

Returns

None.

Description

Invoke **PEXOCCRestoreModelClipVolume** to create an attribute output command which restores the last-saved model clipping volume in the renderer's pipeline state.

This function restores the model clip volume to the value that the structure being traversed inherited. If a root structure is being traversed, then the model clip volume is set to the pipeline state's default *model clip volume*. If you set the model clip indicator (*flag*) to **PEXClip** (see the **PEXOCCSetModelClipFlag** output primitive), then the implementation does not render primitives outside of the model clipping volume.

This function is a member of the global attribute set and affects all output primitives.

Errors

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

See Also

PEXOCCSetModelClipFlag, **PEXOCCSetModelClipVolume**, **PEXOCCSetModelClipVolume2D**

PEXOCCSetAlphaBlend - Set Alpha Blend Function (5.2)

Synopsis

void **PEXOCCSetAlphaBlend**(`PEXOCC` *context*, int *src_blend_func*, int *dest_blend_func*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
src_blend_func	The source blending factor (see Description).
dest_blend_func	The destination blending factor (see Description).

Returns

None.

Description

Invoke **PEXOCCSetAlphaBlend** to create an attribute output command that sets the alpha blend function attribute in the renderer's pipeline state according to the values specified by *src_blend_func* and *dest_blend_func*.

Alpha blending specifies how the implementation should individually modulate the color coming out of the depth cueing stage and the previously rendered pixel. The results are summed to determine a color that the implementation sends through the color approximation stage.

The PEX implementation uses alpha blending if you set the renderer's transparency method to **PEXTransparencyAlphaBlend**. If you set the transparency method to one of the other values, then the implementation ignores this function. If the implementation supports an alpha buffer, then the implementation stores the alpha component of the color directly in the alpha buffer when it sends the other color components through the color approximation stage. Note: the previously rendered pixels have already been through color approximation and the implementation must transform them back into the rendering color model for alpha blending. This can result in some loss of color accuracy. If the color approximation or the rendering color model are different from the values used for the previously rendered pixels, then the transformed color may be incorrect.

PEXlib describes the alpha blending factors and functions according to the following key:

C = Color without its alpha component

A = Alpha component of color

$_{src}$ = Color output from depth cueing

$_{out}$ = Color output from alpha blending

$_{dst}$ = Previously rendered color

In all cases, if C_{out} or A_{out} exceeds 1.0, the value is clamped to 1.0

The overall blend function is determined by combining blend factors:

$$C_{out} = src\_blend\_func * C_{src} + dest\_blend\_func * C_{dst}$$

A_{out} = *src_blend_func* * A_{src} + *dest_blend_func* * A_{dst}

The blend factors *src_blend_func* and *dest_blend_func* are selected from the following table:

Table 7: Alpha Blend Factors

PEXlib Symbol	Value
PEXZero	0
PEXOne	1
PEXSourceColor	C_{src}
PEXDestinationColor	C_{dst}
PEXOneMinusSourceColor	$1 - C_{src}$
PEXOneMinusDestinationColor	$1 - C_{dst}$
PEXSourceAlpha	A_{src}
PEXDestinationAlpha	A_{dst}
PEXOneMinusSourceAlpha	$1 - A_{src}$
PEXOneMinusDestinationAlpha	$1 - A_{dst}$
PEXSourceAlphaMinOneMinusDestination-Alpha	$\min(A_{src}, (1 - A_{dst}))$

The following table lists the alpha blend factor combinations that are defined in PEX. An implementation may not support all of these combinations. Therefore, invoke the **PEXQueryAlphaBlend** function (page 270) to determine if your pair of factors, chosen from the list below, are supported by the implementation.

Table 8: Defined Alpha Blend Functions

Source Blend Factor	Destination Blend Factor	Combined Function
PEXOne	**PEXZero**	$C_{out} = C_{src}$ $A_{out} = A_{src}$
PEXSourceAlpha	**PEXOneMinusSource-Alpha**	$C_{out} = (A_{src} * C_{src}) + ((1-A_{src}) * C_{dst})$ $A_{out} = (A_{src} * A_{src}) + ((1-A_{src}) * A_{dst})$
PEXZero	**PEXZero**	$C_{out} = 0$ $A_{out} = 0$
PEXDestinationColor	**PEXZero**	$C_{out} = C_{src} * C_{dst}$ $A_{out} = A_{src} * A_{dst}$
PEXOneMinusDestination Color	**PEXZero**	$C_{out} = C_{src} * (1 - C_{dst})$ $A_{out} = A_{src} * (1 - A_{dst})$
PEXSourceAlpha	**PEXZero**	$C_{out} = A_{src} * C_{src}$ $A_{out} = A_{src} * A_{src}$

Table 8: Defined Alpha Blend Functions *(Continued)*

Source Blend Factor	Destination Blend Factor	Combined Function
PEXOneMinusSource-Alpha	**PEXZero**	$C_{out} = (1 - A_{src}) * C_{src}$ $A_{out} = (1 - A_{src}) * A_{src}$
PEXDestinationAlpha	**PEXZero**	$C_{out} = A_{dst} * C_{src}$ $A_{out} = A_{dst} * A_{src}$
PEXOneMinusDestination Alpha	**PEXZero**	$C_{out} = (1 - A_{dst}) * C_{src}$ $A_{out} = (1 - A_{dst}) * A_{src}$
PEXZero	**PEXOne**	$C_{out} = C_{dst}$ $A_{out} = A_{dst}$
PEXOne	**PEXOne**	$C_{out} = C_{src} + C_{dst}$ $A_{out} = A_{src} + A_{dst}$
PEXDestinationColor	**PEXOne**	$C_{out} = (C_{src} * C_{dst}) + C_{dst}$ $A_{out} = (A_{src} * A_{dst}) + A_{dst}$
PEXOneMinusDestination Color	**PEXOne**	$C_{out} = (C_{src} * (1 - C_{dst})) + C_{dst}$ $A_{out} = (A_{src} * (1 - A_{dst})) + A_{dst}$
PEXSourceAlpha	**PEXOne**	$C_{out} = (A_{src} * C_{src}) + C_{dst}$ $A_{out} = (A_{src} * A_{src}) + A_{dst}$
PEXOneMinusSource-Alpha	**PEXOne**	$C_{out} = ((1 - A_{src}) * C_{src}) + C_{dst}$ $A_{out} = ((1 - A_{src}) * A_{src}) + A_{dst}$
PEXDestinationAlpha	**PEXOne**	$C_{out} = (A_{dst} * C_{src}) + C_{dst}$ $A_{out} = (A_{dst} * A_{src}) + A_{dst}$
PEXOneMinusDestination Alpha	**PEXOne**	$C_{out} = ((1 - A_{dst}) * C_{src}) + C_{dst}$ $A_{out} = ((1 - A_{dst}) * A_{src}) + A_{dst}$
PEXOne	**PEXSourceColor**	$C_{out} = C_{src} + (C_{src} * C_{dst})$ $A_{out} = A_{src} + (A_{src} * A_{dst})$
PEXOneMinusDestination Color	**PEXSourceColor**	$C_{out} = (C_{src} * (1 - C_{dst})) + (C_{src} * C_{dst})$ $A_{out} = (A_{src} * (1 - A_{dst})) + (A_{src} * A_{dst})$
PEXZero	**PEXOneMinusSource-Color**	$C_{out} = (1 - C_{src}) * C_{dst}$ $A_{out} = (1 - A_{src}) * A_{dst}$
PEXOne	**PEXOneMinusSource-Color**	$C_{out} = C_{src} + ((1 - C_{src}) * C_{dst})$ $A_{out} = A_{src} + ((1 - A_{src}) * A_{dst})$
PEXDestinationColor	**PEXOneMinusSource-Color**	$C_{out} = (C_{src} * C_{dst}) + ((1 - C_{src}) * C_{dst})$ $A_{out} = (A_{src} * A_{dst}) + ((1 - A_{src}) * A_{dst})$
PEXZero	**PEXSourceAlpha**	$C_{out} = A_{src} * C_{dst}$ $A_{out} = A_{src} * A_{dst}$
PEXOne	**PEXSourceAlpha**	$C_{out} = C_{src} + (A_{src} * C_{dst})$ $A_{out} = A_{src} + (A_{src} * A_{dst})$

Table 8: Defined Alpha Blend Functions *(Continued)*

Source Blend Factor	Destination Blend Factor	Combined Function
PEXOneMinusSource-Alpha	**PEXSourceAlpha**	$C_{out} = ((1 - A_{src}) * C_{src}) + (A_{src} * C_{dst})$ $A_{out} = ((1 - A_{src}) * A_{src}) + (A_{src} * A_{dst})$
PEXDestinationAlpha	**PEXSourceAlpha**	$C_{out} = (A_{dst} * C_{src}) + (A_{src} * C_{dst})$ $A_{out} = 2 * (A_{src} * A_{dst})$
PEXOneMinusDestination Alpha	**PEXSourceAlpha**	$C_{out} = ((1 - A_{dst}) * C_{src}) + (A_{src} * C_{dst})$ $A_{out} = ((1 - A_{dst}) * A_{src}) + (A_{src} * A_{dst})$
PEXZero	**PEXOneMinusSource-Alpha**	$C_{out} = (1 - A_{src}) * C_{dst}$ $A_{out} = (1 - A_{src}) * A_{dst}$
PEXOne	**PEXOneMinusSource-Alpha**	$C_{out} = C_{src} + ((1 - A_{src}) * C_{dst})$ $A_{out} = A_{src} + ((1 - A_{src}) * A_{dst})$
PEXDestinationAlpha	**PEXOneMinusSource-Alpha**	$C_{out} = (A_{dst} * C_{src}) + ((1 - A_{src}) * C_{dst})$ $A_{out} = A_{dst}$
PEXOneMinusDestination Alpha	**PEXOneMinusSource-Alpha**	$C_{out} = ((1 - A_{dst}) * C_{src}) + ((1 - A_{src}) * C_{dst})$ $A_{out} = ((1 - A_{dst}) * A_{src}) + ((1 - A_{src}) * A_{dst})$
PEXZero	**PEXDestinationAlpha**	$C_{out} = A_{dst} * C_{dst}$ $A_{out} = A_{dst} * A_{dst}$
PEXOne	**PEXDestinationAlpha**	$C_{out} = C_{src} + (A_{dst} * C_{dst})$ $A_{out} = A_{src} + (A_{dst} * A_{dst})$
PEXSourceAlpha	**PEXDestinationAlpha**	$C_{out} = (A_{src} * C_{src}) + (A_{dst} * C_{dst})$ $A_{out} = (A_{src} * A_{src}) + (A_{dst} * A_{dst})$
PEXOneMinusSource-Alpha	**PEXDestinationAlpha**	$C_{out} = ((1 - A_{src}) * C_{src}) + (A_{dst} * C_{dst})$ $A_{out} = ((1 - A_{src}) * A_{src}) + (A_{dst} * A_{dst})$
PEXOneMinusDestination Alpha	**PEXDestinationAlpha**	$C_{out} = ((1 - A_{dst}) * C_{src}) + (A_{dst} * C_{dst})$ $A_{out} = ((1 - A_{dst}) * A_{src}) + (A_{dst} * A_{dst})$
PEXZero	**PEXOneMinusDestination Alpha**	$C_{out} = (1 - A_{dst}) * C_{dst}$ $A_{out} = (1 - A_{dst}) * A_{dst}$
PEXOne	**PEXOneMinusDestination Alpha**	$C_{out} = C_{src} + ((1 - A_{dst}) * C_{dst})$ $A_{out} = A_{src} + ((1 - A_{dst}) * A_{dst})$
PEXSourceAlpha	**PEXOneMinusDestination Alpha**	$C_{out} = (A_{src} * C_{src}) + ((1 - A_{dst}) * C_{dst})$ $A_{out} = (A_{src} * A_{src}) + ((1 - A_{dst}) * A_{dst})$
PEXOneMinusSource-Alpha	**PEXOneMinusDestination Alpha**	$C_{out} = ((1 - A_{src}) * C_{src}) + ((1 - A_{dst}) * C_{dst})$ $A_{out} = ((1 - A_{src}) * A_{src}) + ((1 - A_{dst}) * A_{dst})$
PEXDestinationAlpha	**PEXOneMinusDestination Alpha**	$C_{out} = (A_{dst} * C_{src}) + ((1 - A_{dst}) * C_{dst})$ $A_{out} = (A_{dst} * A_{src}) + ((1 - A_{dst}) * A_{dst})$
PEXSourceAlphaMinOne MinusDestinationAlpha	**PEXOne**	$C_{out} = (\min(A_{src}, (1 - A_{dst})) * C_{src}) + C_{dst}$ $A_{out} = A_{src} + A_{dst}$

If the implementation does not support the combined alpha blend function specified by the *src_blend_func* and *dest_blend_func* blend factors, then the alpha blend function used by the implementation is the function formed by the source blend factor of **PEXOne** and the destination blend factor of **PEXZero**.

A pixel is transparent if its alpha value is 0.0; it is opaque if its alpha value is 1.0. If you alpha blend a color of a primitive that does not have an alpha component, then the implementation uses the default alpha value. For fill area primitives, the default alpha value is the transmission coefficient field of the pipeline state's reflection attributes subtracted from 1. For back-facing fill area primitives, the pipeline state's back-facing reflection attributes are similarly used. For all other primitives, alpha defaults to 1.0.

Alpha blending interacts with HLHSR, primitive antialiasing, drawing functions (except *GXcopy*), and plane masking (except all planes enabled), in an implementation-dependent way.

This function is a member of the global attribute set and affects all output primitives.

Errors

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

See Also

PEXQueryAlphaBlend

PEXQueryAlphaBlend - Query Support for Alpha Blend Factors (5.2)

Synopsis

int **PEXQueryAlphaBlend**(Display *display*, Drawable *drawable*, int *src_blend_func*, int *dest_blend_func*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	A resource identifier of a drawable.
src_blend_func	The source blending factor (page 265).
dest_blend_func	The destination blending factor (page 265).

Returns

A non-zero value if the implementation supports the combination of the specified alpha blend factors for the specified *drawable*. The function returns zero if the pair is not supported.

Description

Invoke **PEXQueryAlphaBlend** to determine if a pair of alpha blend factors are supported on the specified drawable.

The alpha blend factors are listed in "Alpha Blend Factors" on page 266 and for the list of PEX-defined alpha blend factor combinations, see "Defined Alpha Blend Functions" on page 266.

Errors

BadDrawable	The specified drawable resource identifier is invalid.

See Also

PEXOCCSetAlphaBlend

PEXOCCSetATextAlignment - Set Annotation Text Alignment (5.0)

Synopsis

void **PEXOCCSetATextAlignment**(`PEXOCC` *context*, int *halignment*, int *valignment*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
halignment	The horizontal annotation text alignment (**PEXHAlignNormal**, **PEXHAlignLeft**, **PEXHAlignCenter** or **PEXHAlignRight**).
valignment	The vertical annotation text alignment (**PEXVAlignNormal**, **PEXVAlignTop**, **PEXVAlignCap**, **PEXVAlignHalf**, **PEXVAlignBase** or **PEXVAlignBottom**).

Returns

None.

Description

Invoke **PEXOCCSetATextAlignment** to create an attribute output command which sets the horizontal and vertical annotation text alignment attributes in the renderer's pipeline state according to the values specified by *halignment* and *valignment*.

The annotation text alignment positions the annotation text primitive's extent relative to the text position. The extent is an imaginary parallelogram which has sides that parallel the *x* and *y* axes of the annotation text's coordinate system. The sides are bounded by the characters of the character string according to the font.

PEXlib defines the following horizontal text alignments (*halignment*):

PEXHAlignNormal - aligns the text with the left-side of the character string if you specify an annotation text path of **PEXPathRight**. If you specify an annotation text path of **PEXPathLeft**, then the alignment is based on the right-side of the character string. If you specify an annotation text path of **PEXPathUp** or **PEXPathDown**, then the alignment is based on the center of the character string.**(5.0)**

PEXHAlignLeft - aligns the text whereby the left-side of the extent passes through the text position.**(5.0)**

PEXHAlignCenter - aligns the text whereby the text position lies midway between the left and right sides of the text extent.**(5.0)**

PEXHAlignRight - aligns the text whereby the right-side of the extent passes through the text position.**(5.0)**

PEXlib defines the following vertical text alignments (*valignment*):

PEXVAlignNormal - aligns the text with the base if you specify an annotation text path of

PEXPathRight, **PEXPathLeft**, or **PEXPathUp**. If you specify an annotation text path of **PEXPathDown**, then the alignment is based on the top of the text primitive's extent.**(5.0)**

PEXVAlignTop - aligns the text whereby the top of the extent passes through the text position.**(5.0)**

PEXVAlignCap - aligns the text whereby the capline of the character string passes through the text position if you specify an annotation text path of **PEXPathRight** or **PEXPathLeft**. If you specify an annotation text path of **PEXPathUp** or **PEXPathDown**, then the alignment is based such that the capline of the top-most character in the string passes through the text position.**(5.0)**

The capline is a horizontal line which bounds the top of the characters of the character string according to the font. In general there is space between the capline and the topline for placing additional rendering, such as an accent, above the character string.

PEXVAlignHalf - aligns the text whereby the halfline of the character string passes through the text position if you specify an annotation text path of **PEXPathRight** or **PEXPathLeft**. If you specify a text path of **PEXPathDown** or **PEXPathUp**, then the alignment is based such that the text position lies halfway between the halflines of the top and bottom characters.**(5.0)**

The halfline is a horizontal line which lies halfway between the capline and the baseline according to the font.

PEXVAlignBase - aligns the text whereby the baseline of the string passes through the text position if you specify an annotation text path of **PEXPathRight** or **PEXPathLeft**. If you specify an annotation text path of **PEXPathUp** or **PEXPathDown**, then the alignment is based such that the bottom line of the bottom-most portion of the character in the character string passes through the text position.**(5.0)**

The baseline is a horizontal line upon which the characters sit according to the font. In general there is space between the baseline and the bottomline for placing the lower portions of letters, such as the stems of the lowercase "j" and "q."

PEXVAlignBottom - aligns the text whereby the bottom of the extent passes through the text position.**(5.0)**

If you specify an undefined horizontal or vertical text alignment, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the annotation attribute set and affects all annotation text primitives.

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

PEXOCCSetATextPath

PEXOCCSetATextHeight - Set Annotation Text Height (5.0)

Synopsis

void **PEXOCCSetATextHeight**(`PEXOCC` *context*, double *height*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
height	The annotation text character height.

Returns

None.

Description

Invoke **PEXOCCSetATextHeight** to create an attribute output command which sets the annotation text height attribute in the renderer's pipeline state according to the value specified by *height*.

The annotation text height defines the nominal height of an uppercase character from its baseline to it capline. *Height* is specified in the annotation text primitive's local coordinate system.

> *baseline* - a horizontal line upon which the characters sit according to the font. In general there is space between the baseline and the bottomline for placing the lower portions of letters, such as the stems of the lowercase "j" and "q."
>
> *capline* - a horizontal line which bounds the top of the characters of a character string according to the font. In general there is space between the capline and the topline for placing additional rendering, such as an accent, above the character string.

If the implementation does not support the specified *height* or the computed width, then the implementation maps the height or width to the nearest supported annotation character height or width. You select a font table entry that defines a font group. The supported annotation character height and width depend on the font files you place in this font group. If all of the fonts are stroked fonts, then the implementation supports a continuous range of character sizes.

This function is a member of the annotation attribute set and affects all annotation text primitives.

Errors

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

See Also

PEXOCCSetATextPath - Set Annotation Text Path (5.0)

Synopsis

void **PEXOCCSetATextPath**(`PEXOCC` *context*, int *path*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
path	The text (drawing) path (**PEXPathRight**, **PEXPathLeft**, **PEXPathUp** or **PEXPathDown**).

Returns

None.

Description

Invoke **PEXOCCSetATextPath** to create an attribute output command which sets the annotation text path attribute in the renderer's pipeline state according to the value specified by *path*.

The annotation text path defines the direction in which you want annotation text characters placed. PEXlib defines the following text paths (*path*):

PEXPathRight - places each character after the previous character in a character string along a baseline in the direction of a vector obtained by rotating the character up vector 90 degrees clockwise.**(5.0)**

PEXPathLeft - places each character after the previous character in a character string along a baseline in the opposite direction of a vector obtained by rotating the character up vector 90 degrees clockwise.**(5.0)**

PEXPathUp - places each character after the previous character in a character string in the direction of the character up vector.**(5.0)**

PEXPathDown - places each character after the previous character in a character string in the opposite direction of the character up vector.**(5.0)**

If you do not specify a text *path*, then the path defaults to **PEXPathRight**. If you specify an undefined path, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the annotation attribute set and affects all annotation output primitives.

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

PEXOCCSetATextAlignment, PEXOCCSetATextUpVector

PEXOCCSetATextStyle - Set Annotation Text Style (5.0)

Synopsis

void **PEXOCCSetATextStyle**(`PEXOCC` *context*, int *style*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
style	The annotation text style (**PEXATextNotConnected** or **PEXATextConnected**).

Returns

None.

Description

Invoke **PEXOCCSetATextStyle** to create an attribute output command which sets the annotation text style attribute in the renderer's pipeline state according to the value specified by *style*.

The annotation text style describes if and how you want annotation text offset rendered. PEXlib defines the following annotation text styles (*style*):

> **PEXATextNotConnected** - no line connects the annotation point to the reference point of the annotation text primitive.**(5.0)**
>
> **PEXATextConnected** - a line connects the annotation to the reference points of the annotation text primitive. PEXlib uses the current line attributes to render this line.**(5.0)**

If the implementation does not support the specified *style*, then the style defaults to **PEXATextNotConnected**. Inquire the supported values for the annotation text style by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the annotation attribute set and affects all annotation text primitives.

Errors

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

See Also

PEXGetEnumTypeInfo

PEXOCCSetATextUpVector - Set Annotation Text Up Vector (5.0)

Synopsis

void **PEXOCCSetATextUpVector**(`PEXOCC` *context*, `PEXVector2D` **vector*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
vector	The annotation text character up vector [§2.31.2, page 67].

Returns

None.

Description

Invoke **PEXOCCSetATextUpVector** to create an attribute output command which sets the annotation text up vector attribute in the renderer's pipeline state according to the value specified by *vector*.

The annotation text up vector defines the up direction for characters. Specify *vector* in text local coordinates. The implementation only applies the direction of the vector, not the magnitude.

If the annotation text up *vector* you specify is degenerate (e.g., has a length of zero), then the vector defaults to a value of <0,1>.

This function is a member of the annotation attribute set and affects all annotation text output primitives.

Errors

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

See Also

PEXOCCSetBFDataMappingIndex - Set Back-Facing Data Mapping Index (5.2)

Synopsis

void **PEXOCCSetBFDataMappingIndex**(`PEXOCC` *context*, unsigned int *index*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The index into the data mapping table.

Returns

None.

Description

Invoke **PEXOCCSetBFDataMappingIndex** to create an attribute output command which sets the back-facing data mapping index attribute in the renderer’s pipeline state according to the value specified by *index*.

You use the data mapping index to select an entry of the renderer’s data mapping table. This entry supplies the data mapping method when you set the data mapping method attribute’s ASF to **PEXBundled** (see **PEXOCCSetIndividualASF**).

To apply the back-facing data mapping index to the back-facing surfaces of output primitives which can be data mapped, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the data mapping index when rendering back-facing surfaces (see **PEXOCCSetDataMappingMethod**).

If you specify an *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the data mapping attribute set and affects all output primitives which can be data mapped.

Errors

BadPEXOutputCommand	The index is zero.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCSetDataMappingIndex, **PEXOCCSetIndividualASF**

PEXOCCSetBFDataMappingMethod - Set Back-Facing Data Mapping Method (5.2)

Synopsis

void **PEXOCCSetBFDataMappingMethod**(`PEXOCC` *context*, int *method*, unsigned int *data_rec_len*, `PEXPointer` *data_rec*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
method	The data mapping method (**PEXDataMapNone** or **PEXDataMapColor**)
data_rec_len	The length of the supplied data record.
data_rec	A pointer to the data record [§2.18.1, page 46].

Returns

None.

Description

Invoke **PEXOCCSetBFDataMappingMethod** to create an attribute output command which sets the back-facing data mapping method in the renderer's pipeline state according to the value specified by *method*.

The data mapping method specifies the mapping of color, or other application-specific data, to intrinsic color. This is the first stage in the color pipeline.

By setting the back-facing data mapping method attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's back-facing data mapping method entry to define the method for mapping color, or other application-specific data, to intrinsic color. When you set the back-facing data mapping method attribute's ASF to **PEXBundled**, the implementation uses the back-facing data mapping method from the data mapping table.

To apply the back-facing data mapping method to the back-facing surfaces of output primitives, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the data mapping method when rendering back-facing surfaces (see **PEXOCCSetDataMappingMethod**).

PEXlib defines the following back-facing data mapping methods (*method*):

> **PEXDataMapNone** - prioritizes the source of the intrinsic color, from first to last, as vertex color (if defined with the primitive), facet color (if defined with the primitive), and surface color. This method does not require any data.**(5.2)**
>
> **PEXDataMapColor** - you define a prioritized list, which includes at least one of three sources. The source of intrinsic color is the vertex color (if defined with the primitive), facet color (if defined with the primitive), or surface color. The prioritized list starts with your top choice. If there is no match between the prioritized list of source selectors and the data defined with a primitive being data mapped, or if the source selectors are otherwise invalid, then the surface color is used as the source.**(5.2)**

If the implementation does not support the specified data mapping method, then the method defaults to **-PEXDataMapNone**.

If the data record size (*data_rec_len*) does not match the data mapping method (*method*) or if you specify an undefined source selector, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the data mapping attribute set and affects all output primitives which can be data mapped.

Data Structures

The method **PEXDataMapNone** requires no data record.

The method **PEXDataMapColor** requires a list of source selectors chosen from the set (**PEXDMMSurfaceColor, PEXDMMVertexColor, PEXDMMFacetColor**). See "PEXColorApproxEntry Data Structure" on page 39.

Errors

BadPEXOutputCommand	The data record is incorrect.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCSetDataMappingMethod, PEXOCCSetFacetDistinguishFlag, PEXOCCSetIndividualASF

PEXOCCSetBFInteriorBundleIndex - Set Back-Facing Interior Bundle Index (5.2)

Synopsis

void **PEXOCCSetBFInteriorBundleIndex**(PEXOCC *context*, unsigned int *index*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The index into the interior bundle table.

Returns

None.

Description

Invoke **PEXOCCSetBFInteriorBundleIndex** to create an attribute output command which sets the back-facing interior bundle index attribute in the renderer's pipeline state according to the value specified by *index*.

You use the back-facing interior bundle index to select an entry of the renderer's interior bundle table whenever a bundlable back-facing interior attribute's ASF is set to **PEXBundled** (see **PEXOCCSetIndividualASF**) and you have set the renderer's PHIGS interior bundle table to **True**. PEXlib supports the bundling of the following back-facing interior attributes: back-facing surface interior style, back-facing surface interior style index, back-facing surface color, and back-facing surface interpolation method.

To apply the back-facing attributes to the back-facing interiors of output primitives, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the interior bundle index (see **PEXOCCSetInteriorBundleIndex**) and ignores the back-facing interior bundle index.

If you specify an undefined interior bundle *index*, then the index defaults to index one. If index one is not defined, then the implementation sets the bundled back-facing surface interior style to **PEXInteriorStyleHollow**, back-facing interior style index to 1, back-facing surface color to index color 1, and back-facing surface interpolation method to **PEXSurfaceInterpNone**. If you specify an interior bundle *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

See also the discussion on PHIGS interior bundle mode (page 450).

This function is a member of the styled and shaded surface attribute set.

Errors

BadPEXOutputCommand	The index is zero.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCSetFacetDistinguishFlag, PEXOCCSetIndividualASF, PEXOCCSetInteriorBundleIndex

PEXOCCSetBFInteriorStyle - Set Back-Facing Surface Interior Style (5.0)

Synopsis

void **PEXOCCSetBFInteriorStyle**(`PEXOCC` *context*, int *style*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
style	The back-facing surface interior style (**PEXInteriorStyleHollow**, **PEXInteriorStyleSolid**, **PEXInteriorStylePattern**, **PEXInteriorStyleHatch** or **PEXInteriorStyleEmpty**).

Returns

None.

Description

Invoke **PEXOCCSetBFInteriorStyle** to create an attribute output command which sets the back-facing surface interior style attribute in the renderer's pipeline state according to the value specified by *style*.

By setting the back-facing interior style attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's back-facing surface interior style entry to describe how to fill the interior of back-facing surfaces. When you set the back-facing interior style attribute's ASF to **PEXBundled**, the implementation uses the back-facing interior style value from the interior bundle table.

To apply the back-facing interior style to the back-facing surfaces of output primitives, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the interior style when rendering back-facing surfaces (see **PEXOCCSetInteriorStyle**).

PEXlib defines the following interior styles (*style*):

PEXInteriorStyleHollow - draws the boundary of the back-facing surfaces of output primitives using the current back-facing surface color but does not fill the interior.**(5.0)**

PEXInteriorStyleSolid - fills the interior of back-facing surfaces of output primitives using the current back-facing surface color.**(5.0)**

PEXInteriorStylePattern - fills the interior of back-facing surfaces of output primitives using the pattern defined according to the entry of the renderer's pattern table specified by the back-facing interior style index.**(5.0)**

PEXInteriorStyleHatch - fills the interior of back-facing surfaces of output primitives using the current surface color and the hatch style specified by the back-facing interior style index.**(5.0)**

PEXInteriorStyleEmpty - does not draw the interiors of the back-facing surfaces of output primitives at all.**(5.0)**

PEXInteriorStyleTexture - textures the interior of the back-facing surfaces of output primitives using the texture maps specified by the back-facing active textures. **(5.2)**

If you do not specify an interior style that is supported by the implementation, then the style defaults to **PEXInteriorStyleHollow**. Inquire the supported values for the back-facing surface interior styles by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the styled and shaded surface attribute set.

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, **PEXOCCSetFacetDistinguishFlag, PEXOCCSetIndividualASF**, **PEXOC-CSetInteriorStyle**

PEXOCCSetBFInteriorStyleIndex - Set Back-Facing Surface Interior Style Index (5.0)

Synopsis

void **PEXOCCSetBFInteriorStyleIndex**(`PEXOCC` *context*, int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The back-facing surface interior style index.

Returns

None.

Description

Invoke **PEXOCCSetBFInteriorStyleIndex** to create an attribute output command which sets the back-facing interior style index attribute in the renderer's pipeline state according to the value specified by *index*.

By setting the back-facing interior style index attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's back-facing surface interior style index to select an entry of the renderer's interior bundle table. When you set the back-facing interior style index attribute's ASF to **PEXBundled**, the implementation uses the back-facing interior style index from the interior bundle table.

To apply the back-facing interior style index to the back-facing surfaces of output primitives, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the interior style index when rendering back-facing surfaces (see **PEXOCCSetInterior-StyleIndex**).

If you set the current interior style to **PEXInteriorStylePattern** or **PEXInteriorStyleHatch**, then the implementation uses back-facing interior surface index to further define the rendering style of back-facing surface primitives. If you set the current interior style to **PEXInteriorStylePattern**, without defining the specified pattern table index, then the index defaults to a table index of one.

For **PEXInteriorStyleHatch**, *index* determines the hatch style. PEXlib defines the following hatch styles:

> **PEXInteriorStyleHatch45Degrees** - draws the back-facing surface interior with single pixel width lines at about 45 degrees, in device coordinates.**(5.2)**
>
> **PEXInteriorStyleHatch135Degrees** - draws the back-facing surface interior with single pixel width lines at about 135 degrees, in device coordinates.**(5.2)**

If the implementation does not support the value specified in *index*, then the hatch style defaults to **PEXInteriorStyleHatch45Degrees**. Inquire the supported values for the hatch style by invoking the **PEXGetEnumTypeInfo** function.

For **PEXInteriorStylePattern**, *index* specifies the index into the Pattern LUT. If the Pattern LUT entry specified by *index* is zero, the implementation uses an index of 1. For most other LUT index setting output commands, attempting to set an unsupported index, such as index 0 in a Pattern LUT, results in an Output Command error. In this case, the current pipeline state determines whether the index is used for an Interior Style index or a Pattern LUT index. Since the pipeline state is not available for an OC stored in a structure until the structure is rendered, the criteria for generating an OC error is not known until the structure is rendered. Since the PEX implementation cannot report OC errors while rendering structures, this OC therefore cannot generate OC errors; it falls back to index 1 instead.

This function is a member of the styled and shaded surface attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, **PEXOCCSetFacetDistinguishFlag, PEXOCCSetBFInteriorStyle, PEXOCCSetIndividualASF**, **PEXOCCSetInteriorStyleIndex**

PEXOCCSetBFReflectanceBundleIndex - Set Back-Facing Reflectance Bundle Index (5.2)

Synopsis

void **PEXOCCSetBFReflectanceBundleIndex**(`PEXOCC` *context*, unsigned int *index*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The index into the interior bundle table.

Returns

None.

Description

Invoke **PEXOCCSetBFReflectanceBundleIndex** to create an attribute output command which sets the back-facing reflectance bundle index attribute in the renderer's pipeline state according to the value specified by *index*.

You use the back-facing reflectance bundle index to select an entry of the renderer's interior bundle table whenever a bundlable back-facing reflectance attribute's ASF is set to **PEXBundled** (see **PEXOCCSetIndividualASF**) and you have set the renderer's PHIGS interior bundle table to **True**. If you set the PHIGS interior bundle mode to **False**, then the implementation uses the back-facing interior bundle index instead and ignores the back-facing reflectance bundle index.

To apply the back-facing attributes to the back-facing interiors of output primitives, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the interior bundle index (see **PEXOCCSetReflectanceBundleIndex**) and ignores the back-facing interior bundle index.

If you specify a back-facing reflectance bundle *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the general surface attribute set.

Errors

BadPEXOutputCommand	The index is zero.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCSetFacetDistinguishFlag, PEXOCCSetIndividualASF, PEXOCCInteriorBundleIndex, PEXOCCSetReflectanceBundleIndex

PEXOCCSetBFReflectionAttributes - Set Back-Facing Surface Reflection Attributes (5.0)

Synopsis

void **PEXOCCSetBFReflectionAttributes**(`PEXOCC` *context*, `PEXReflectionAttributes5_2` **attributes*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
attributes	The back-facing surface reflection attributes [§2.24.11, page 59], [§2.24.10, page 58].

Returns

None.

Description

Invoke **PEXOCCSetBFReflectionAttributes** to create an attribute output command which sets the back-facing surface reflection attributes in the renderer's pipeline state according to the values specified by *attributes*.

By setting the back-facing reflection attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's back-facing surface reflection attributes entry to define the reflection calculation used to light the back-facing surfaces of output primitives. When you set the back-facing reflection attribute's ASF to **PEXBundled**, the implementation uses the back-facing reflection attributes values from the interior bundle table.

To apply the back-facing reflection attributes to the back-facing surfaces of output primitives, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the reflection attributes when rendering back-facing surfaces (see **PEXOCCSetReflectionAttributes**).

The *attributes* argument defines surface reflection attributes that include ambient, diffuse and specular coefficients, the specular concentration and color, and the transmission coefficient. The ambient, diffuse, and specular coefficients define the fraction of ambient, diffuse, and specular light respectively reflected by a back-facing surface. Specify a value of 0.0 to reflect no light. Specify a value of 1.0 to reflect 100% of the light.

The larger the value of the *specular concentration* value of the back-facing reflection attributes, the shinier the back-facing surface. If the implementation does not support the specified color type, then the implementation uses a back-facing specular color type of **PEXColorTypeIndexed** with a value of one. For out of range direct color components, the result is implementation-dependent. If you specify an indexed color with an index of 65535, then the implementation issues a *BadPEXOutputCommand* error.

Inquire whether the transmission coefficient is used in the reflection calculation by inquiring the transparency supported via the **PEXGetImpDepConstants** function. The transmission coefficient defines the default alpha component of back-facing surface primitives without an explicit alpha-component. Specify a transmission coefficient of 0.0 for an opaque surface. Specify a value of 1.0 for a transparent surface. The alpha-component value is determined by subtracting the transmission coefficient from 1.

The back-facing surface reflection attribute is a member of the general surface attribute set.

Errors

BadPEXOutputCommand	The specified color index is 65535.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXGetImpDepConstants, PEXOCCSetFacetDistinguishFlag, PEXOCCSetIndividualASF, **PEXOCCSetReflectionAttributes**

PEXOCCSetBFReflectionModel - Set Back-Facing Surface Reflection Model (5.0)

Synopsis

void **PEXOCCSetBFReflectionModel**(PEXOCC *context*, int *model*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
model	The back-facing surface reflection model (**PEXRMNone**, **PEXRMAmbient**, **PEXRMAmbientDiffuse** or **PEXRMAmbientDiffuseSpecular**).

Returns

None.

Description

Invoke **PEXOCCSetBFReflectionModel** to create an attribute output command which sets the back-facing surface reflection model attribute in the renderer's pipeline state according to the value specified by *model*.

By setting the back-facing reflection model attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's back-facing surface reflection model entry to specify which reflectance components to consider when performing reflectance calculations on back-facing surfaces of output primitives. When you set the back-facing reflection model attribute's ASF to **PEXBundled**, the implementation uses the back-facing reflection model value from the interior bundle table.

To apply the back-facing reflection model to the back-facing surfaces of output primitives, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the reflection model when lighting back-facing surfaces (see **PEXOCCSetReflectionModel**).

PEXlib defines the following reflection models (*model*):

PEXRMNone - performs no reflectance calculation. The back-facing surface color is not affected by light sources.**(5.2)**

PEXRMAmbient - uses only the ambient terms of the reflectance calculation.**(5.2)**

PEXRMAmbientDiffuse - uses only the ambient and diffuse terms of the reflectance calculation.**(5.2)**

PEXRMAmbientDiffuseSpecular - uses the ambient, diffuse, and specular terms of the reflectance calculation.**(5.2)**

Note: The reflection models defined above simply have new names for PEXlib 5.2 and are completely equivalent to their respective names in PEXlib 5.1: **PEXReflectionNone**, **PEXReflectionAmbient**,

PEXReflectionDiffuse, and **PEXReflectionSpecular**

If the implementation does not support the specified reflection *model*, then the model defaults to **PEXRMNone**. Inquire the supported values for the surface reflection model by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the general surface attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, **PEXOCCSetFacetDistinguishFlag, PEXOCCSetIndividualASF**, **PEXOCCSetReflectionModel**

PEXOCCSetBFSurfaceActiveTextures - Set Back-Facing Surface Active Textures (5.2)

Synopsis

void **PEXOCCSetBFSurfaceActiveTextures**(`PEXOCC` *context*, unsigned int *count*, `PEXTableIndex` **indices*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
count	The number of indices in the list.
indices	A list of table indices into the texture mapping lookup table [§2.3.21, page 29].

Returns

None.

Description

Invoke **PEXOCCSetBFSurfaceActiveTextures** to create an attribute output command which sets the back-facing surface active textures attribute in the renderer's pipeline state according to the values specified by *indices*.

You use the pipeline state's active texture indices as an entry into the renderer's texture mapping lookup table. The indices are an ordered list that you use to select the textures that are sequentially applied to textured surfaces. The active texture specifies the texture map and an associated set of values. It applies only to surface primitives when the primitive's interior style is **PEXInteriorStyleTexture** and the active textures field is non-empty.

To apply the back-facing active texture to the back-facing surfaces of textured output primitives, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the surface active textures to apply textures to back-facing surfaces (see **PEXOCCSetSurfaceActiveTextures**).

The indices are inserted into the renderer's texture mapping lookup table. If the number of indices you specify exceeds the maximum number of texture maps that your implementation simultaneously supports, then the implementation ignores textures defined beyond the limit. Inquire the maximum number of texture maps, which you can apply to a single surface, by invoking the **PEXGetImpDepConstants** function.

If you specify a texture mapping lookup table *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the styled and shaded surface attribute set.

Errors

BadPEXOutputCommand	An index is zero.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXGetImpDepConstants, PEXOCCSetFacetDistinguishFlag, PEXOCCSetSurfaceActiveTextures

PEXOCCSetBFSurfaceColor - Set Back-Facing Surface Color (5.0)

Synopsis

void **PEXOCCSetBFSurfaceColor**(`PEXOCC` *context*, `PEXColor` **color*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
color_type (OCC)	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
color	A pointer to the back-facing surface color [§2.5.12, page 32].

Returns

None.

Description

Invoke **PEXOCCSetBFSurfaceColor** to create an attribute output command which sets the back-facing surface color attribute in the renderer’s pipeline state according to the value specified by *color*.

By setting the back-facing surface color attribute’s ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state’s back-facing surface color entry to select the color for back-facing surfaces. When you set the back-facing surface color attribute’s ASF to **PEXBundled**, the implementation uses the back-facing surface color from the interior bundle table.

To apply the back-facing surface color to the back-facing surfaces of output primitives, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the surface color when applying color to back-facing surfaces (see **PEXOCCSetSurfaceColor**).

Depending on the *color type* you specify, this function sets the attribute to either an indexed color or a direct color value. If you specify a *color type* of **PEXColorTypeIndexed**, but you specified an undefined index, then the index defaults to color index one. If color index one is not defined, then the back-facing surface color defaults to white. If you specified a color index greater than 65534, then the implementation issues a *BadPEXOutputCommand* error. If the *color type* is one of the supported direct color types and you specify an out of range component, then the result is implementation-dependent. If the *color type* is not supported, then indexed color one is used. Inquire the supported values for *color type* by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the styled and shaded surface attribute set.

Errors

BadPEXOutputCommand 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

PEXCreateLookupTable, PEXGetEnumTypeInfo, PEXOCCSetFacetDistinguishFlag, PEXOCCSetIndividualASF, **PEXOCCSetSurfaceColor**

PEXOCCSetBFSurfaceColorIndex - Set Back-Facing Surface Color Index(5.0)

Synopsis

void **PEXOCCSetBFSurfaceColorIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The color table index for back-facing surfaces.

Returns

None.

Description

Invoke **PEXOCCSetBFSurfaceColorIndex** to set the back-facing surface color attribute to an indexed color in the renderer's pipeline state according to the value specified by *index*. The index selects an entry of the renderer's color table. You can alternately use the **PEXOCCSetBFSurfaceColor** function which allows you to specify a color of any type.

By setting the back-facing surface color attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's surface color entry to determine the color of back-facing surface attributes. When you set the back-facing surface color attribute's ASF to **PEXBundled**, the implementation uses the surface color from the interior bundle table.

To apply the back-facing surface color index to the back-facing surfaces of output primitives, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the surface color index when rendering back-facing surfaces (see **PEXOCCSetSurfaceColorIndex**).

If you specify an undefined color table *index*, then the index defaults to one. If color table index one is not defined, then the back-facing surface color defaults to white. If you specify a color table *index* which is greater than 65534, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the styled and shaded surface attribute set.

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

PEXCreateLookupTable, PEXOCCSetBFSurfaceColor, PEXOCCSetFacetDistinguishFlag, PEXOCCSetIndividualASF, PEXOCCSetSurfaceColorIndex

PEXOCCSetBFSurfaceInterpMethod - Set Back-Facing Surface Interpolation Method (5.0)

Synopsis

void **PEXOCCSetBFSurfaceInterpMethod**(`PEXOCC` *context*, int *method*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
method	The back-facing surface interpolation method (**PEXSurfaceInterpNone**, **PEXSurfaceInterpColor**, **PEXSurfaceInterpDotProduct** or **PEXSurfaceInterpNormal**).

Returns

None.

Description

Invoke **PEXOCCSetBFSurfaceInterpMethod** to create an attribute output command which sets the back-facing surface interpolation method attribute in the renderer's pipeline state according to the value specified by *method*.

By setting the back-facing surface interpolation method attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the back-facing surface interpolation method entry to specify how color values, including any alpha component, are computed in the interior of back-facing surface primitives. When you set the back-facing surface interpolation method attribute's ASF to **PEXBundled**, the implementation uses the back-facing surface interpolation method from the interior bundle table.

To apply the back-facing interpolation method to the back-facing surfaces of output primitives, you must set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**). If you set the flag to **False**, then the implementation uses the surface interpolation method when applying the interpolation method to back-facing surfaces (see **PEXOCCSetSurfaceInterpMethod**).

PEXlib defines the following interpolation methods (*method*):

> **PEXSurfaceInterpNone** - uses the color resulting from a single evaluation of the reflectance calculation to render the entire back-facing surface. Interpolation is *not* performed across surface interiors or edges.**(5.0)**
>
> **PEXSurfaceInterpColor** - performs the reflection calculation at the vertices of the back-facing surface. Then linearly interpolates the resulting color values across the interior of the surface and the edges.**(5.0)**
>
> **PEXSurfaceInterpDotProduct** - computes at each vertex any dot products needed by the reflectance calculation. Then linearly interpolates these dot products and performs the reflectance calculation

using these values to compute the color value at each pixel in the interior of a back-facing surface or along a back-facing surface edge.**(5.0)**

PEXSurfaceInterpNormal - interpolates vertex reflectance normals and performs the reflectance calculation at each pixel in the interior of a back-facing surface or along a back-facing surface edge.**(5.0)**

If the implementation does not support the specified interpolation *method*, then the method defaults to **PEXSurfaceInterpNone**. Inquire the supported values for the back-facing surface interpolation *method* by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the styled and shaded surface attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, **PEXOCCSetFacetDistinguishFlag, PEXOCCSetIndividualASF**, **PEXOCCSetSurfaceInterpMethod**

PEXOCCSetCharExpansion - Set Character Expansion Factor (5.0)

Synopsis

void **PEXOCCSetCharExpansion**(`PEXOCC` *context*, double *expansion*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
expansion	The character expansion factor.

Returns

None.

Description

Invoke **PEXOCCSetCharExpansion** to create an attribute output command which sets the character expansion factor attribute in the renderer's pipeline state according to the value specified by *expansion*.

By setting the character expansion attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's character expansion entry to scale the character width: character width is multiplied by the magnitude of the character expansion. The character height (see **PEXOCCSetCharHeight**) and the text font index (see **PEXOCCSetTextFontIndex**) determine the character width because each font includes a width:height ratio. When you set the character expansion attribute's ASF to **PEXBundled**, the implementation uses the character expansion factor from the text bundle table.

PEXlib compares the character expansion factor that you specify to the minimum and maximum character expansion factors. You select a font table entry that defines a font group. The minimum and maximum character expansion factor values depend on the font files that you place in the font group. If the font is a stroked font, then a continuous range of character widths are supported. Otherwise PEXlib uses the closest width supported by the font. For example, if your application opens all of the stroke fonts that are possible to scale and rotate, then the implementation supports a continuous number of expansions. If you specify an expansion that is smaller than the minimum character expansion factor, then the expansion factor defaults to the minimum value. If the expansion is larger than the maximum character expansion factor, then the expansion factor defaults to the maximum value.

This function is a member of the annotation and text attribute set and affects all annotation text and text primitives.

Errors

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

See Also

PEXOCCSetCharHeight, PEXOCCSetIndividualASF, **PEXOCCSetTextFontIndex**

PEXOCCSetCharHeight - Set Character Height (5.0)

Synopsis

void **PEXOCCSetCharHeight**(PEXOCC *context*, double *height*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
height	The text character height.

Returns

None.

Description

Invoke **PEXOCCSetCharHeight** to create an attribute output command which sets the character height value attribute in the renderer's pipeline state according to the value specified by *height*.

Text height defines the nominal height of an uppercase character from its baseline to its capline.

> *baseline* - a horizontal line upon which the characters sit according to the font. In general there is space between the baseline and the bottomline for placing the lower portions of letters, such as the stems of the lowercase "j" and "q."
>
> *capline* - a horizontal line which bounds the top of the characters of a character string according to the font. In general there is space between the capline and the topline for placing additional rendering, such as an accent, above the character string.

The character height, along with the text font index (see **PEXOCCSetTextFontIndex**), determine the character width because each font includes a width:height ratio. The supported heights and widths depend on the font files you place in the font group defined by the font table entry which you select. If the font is a stroked font, then a continuous range of character heights and widths are supported. Otherwise PEXlib uses the closest values supported by the font.

Specify *height* in the text local coordinate system.

This function is a member of the text attribute set and affects text primitives.

Errors

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

See Also

PEXOCCSetCharExpansion, **PEXOCCSetTextFontIndex**

PEXOCCSetCharSpacing - Set Character Spacing (5.0)

Synopsis

void **PEXOCCSetCharSpacing**(`PEXOCC` *context*, double *spacing*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
spacing	The text character spacing factor.

Returns

None.

Description

Invoke **PEXOCCSetCharSpacing** to create an attribute output command which sets the character spacing attribute in the renderer’s pipeline state according to the value specified by *spacing*.

By setting the character spacing attribute’s ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state’s character spacing entry to define the space between adjacent characters in a character string. Express the character spacing attribute (*spacing*) as a fraction of the character height (see **PEXOCCSetCharHeight**). To overlap characters, specify a negative value for *spacing*. When you set the character spacing attribute’s ASF to **PEXBundled**, the implementation uses the character spacing from the text bundle table.

This function is a member of the annotation and text attribute set and affects text primitives.

Errors

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

See Also

PEXOCCSetCharHeight, PEXOCCSetIndividualASF

PEXOCCSetCharUpVector - Set Character Up Vector (5.0)

Synopsis

void **PEXOCCSetCharUpVector**(`PEXOCC` *context*, `PEXVector2D` **vector*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
vector	The text character up vector [§2.31.2, page 67].

Returns

None.

Description

Invoke **PEXOCCSetCharUpVector** to create an attribute output command which sets the text up vector attribute in the renderer's pipeline state according to the value specified by *vector*.

Use the character up *vector* to define the up direction for characters in the text plane.

Specify *vector* in text local coordinates. The direction vectors associated with the text primitive define this local coordinate system.

If the text up *vector* you specify is degenerate (e.g., it has a length of zero), then the vector defaults to a value of <0,1>.

This function is a member of the text attribute set and affects text primitives.

Errors

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

See Also

PEXOCCSetColorApproxIndex - Set Color Approximation Index (5.0)

Synopsis

void **PEXOCCSetColorApproxIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The color approximation table index.

Returns

None.

Description

Invoke **PEXOCCSetColorApproxIndex** to create an attribute output command which sets the color approximation index attribute in the renderer's pipeline state according to the value specified by *index*.

Use color approximation *index* to select an entry of the renderer's color approximation table. Color approximation defines how, at the end of the rendering pipeline, colors (except their alpha components) are transformed into pixel values.

If you specify an undefined color *index*, then the index defaults to color index zero. If the *index* you specify exceeds 65534, then the implementation issues a *BadPEXOutputCommand* error.

The color approximation function is a member of the global attribute set.

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

PEXOCCSetCurveApprox - Set Curve Approximation Method (5.0)

Synopsis

void **PEXOCCSetCurveApprox**(`PEXOCC` *context*, int *method*, double *tolerance*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
method	The curve approximation method (**PEXApproxImpDep**, **PEXApproxConstantBetweenKnots**, **PEXApproxWCChordalSize**, **PEXApproxNPCChordalSize**, **PEXApproxDCChordalSize**, **PEXApproxWCChordalDev**, **PEXApproxNPCChordalDev**, **PEXApproxDCChordalDev**, **PEXApproxWCRelative, PEXApproxNPCRelative** or **PEXApproxDCRelative**).
tolerance	The curve approximation tolerance (specific to each method).

Returns

None.

Description

Invoke **PEXOCCSetCurveApprox** to create an attribute output command which sets the curve approximation attribute in the renderer's pipeline state according to the value specified by *method*.

By setting the curve approximation attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's curve approximation entry to specify the smoothing method for the rendering of curved univariate primitives. When you set the curve approximation attribute's ASF to **PEXBundled**, the implementation uses the curve approximation method from the line bundle table.

PEXlib defines the following curve approximation methods (*method*) and the use of *tolerance*:

PEXApproxImpDep - smooths the curve in an implementation-dependent way and ignores *tolerance*.**(5.0)**

PEXApproxConstantBetweenKnots - segments the curve in equal parametric increments between successive pairs of knots. If there are no explicit knots, then the curve is treated as having a single knot. If the primitive is a circular or elliptical arc, then the circle or ellipse from which it is formed is considered to have a single knot.**(5.0)**

Tolerance controls the segmentation. If you specify a fraction for *tolerance*, then the implementation truncates it and only uses the integer portion. If you specify a *tolerance* that is less than or equal to zero, then the curve is evaluated only at the parameter limits and at the knots that are within the specified parameter range. If *tolerance* is greater than zero, then the curve is evaluated at the parameter limits, at the knots that are within the specified parameter range, and at the number of positions specified by tolerance between each pair of knots.

PEXApproxWCChordalSize - segments the curve whereby the WC length of each segment (chord) is less than the *tolerance* (specified in world coordinates).**(5.0)**

PEXApproxNPCChordalSize - segments the curve whereby the NPC length of each segment (chord) is less than the *tolerance* (specified in normalized projection coordinates).**(5.0)**

PEXApproxDCChordalSize - segments the curve whereby the DC length of each segment (chord) is less than the *tolerance* (specified in device coordinates).**(5.0)**

PEXApproxWCChordalDev - segments the curve whereby the deviation in WC's between the curve and its segmented representation does not exceed the *tolerance* (specified in world coordinates).**(5.0)**

PEXApproxNPCChordalDev - segments the curve whereby the deviation in NPC's between the curve and its segmented representation does not exceed the *tolerance* (specified in normalized projection coordinates).**(5.0)**

PEXApproxDCChordalDev - segments the curve whereby the deviation in DC's between the curve and its segmented representation does not exceed the *tolerance* (specified in device coordinates).**(5.0)**

PEXApproxWCRelative - indicates, along with a *tolerance* value in the range [0.0, 1.0], a relative quality of rendering to be maintained independent of scaling in world coordinates (WC's). Specify a *tolerance* of 1.0 for the highest possible quality.**(5.0)**

PEXApproxNPCRelative - indicates, along with a *tolerance* value in the range [0.0, 1.0], a relative quality of rendering to be maintained independent of scaling in normalized projection coordinates (NPC's). Specify a *tolerance* of 1.0 for the highest possible quality.**(5.0)**

PEXApproxDCRelative - indicates, along with a *tolerance* value in the range [0.0, 1.0], a relative quality of rendering to be maintained independent of scaling in device coordinates (DC's). Specify a *tolerance* of 1.0 for the highest possible quality.**(5.0)**

If the implementation does not support the specified method, then the method defaults to **PEXApproxImpDep**. Inquire the supported values for the curve approximation by invoking the **PEXGetEnumTypeInfo** function.

Specify the *tolerance* argument to indicate the desired accuracy of the approximation. The curve approximation tolerance has different results depending on which method you specify. In general, specify a lower relative quality to smooth the curved univariate primitives at a higher speed. However, implementations may smooth the curves at a higher quality than you requested.

This function is a member of the curve attribute set and affects curved primitives.

Errors

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

See Also

PEXGetEnumTypeInfo, **PEXOCCSetIndividualASF**

PEXOCCSetDataMappingIndex - Set Data Mapping Index (5.2)

Synopsis

void **PEXOCCSetDataMappingIndex**(`PEXOCC` *context*, unsigned int *index*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The index into the data mapping table.

Returns

None.

Description

Invoke **PEXOCCSetDataMappingIndex** to create an attribute output command which sets the data mapping index attribute in the renderer's pipeline state according to the value specified by *index*.

You use the data mapping index to select an entry of the renderer's data mapping table. This entry supplies the data mapping method when you set the data mapping method attribute's ASF to **PEXBundled** (see **PEXOCCSetIndividualASF**).

If you do not set the facet distinguish mode to **True** (see **PEXOCCSetFacetDistinguishFlag**), then the implementation applies this data mapping index to back-facing as well as to front-facing surfaces. Otherwise the implementation uses the back-facing data mapping index.

If you specify an *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the data mapping attribute set.

Errors

BadPEXOutputCommand	The index is zero.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCSetBFDataMappingIndex, PEXOCCSetFacetDistinguishFlag, PEXOCCSetIndividualASF

PEXOCCSetDataMappingMethod - Set Data Mapping Method (5.2)

Synopsis

void **PEXOCCSetDataMappingMethod**(`PEXOCC` *context*, int *method*, unsigned int *data_rec_len*,
`PEXPointer` *data_rec*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
method	The data mapping method (**PEXDataMapNone** or **PEXDataMapColor**)
data_rec_len	The length of the supplied data record.
data_rec	A pointer to the data record [§2.18.1, page 46].

Returns

None.

Description

Invoke **PEXOCCSetDataMappingMethod** to create an output primitive which sets the data mapping method in the renderer's pipeline state according to the value specified by *method*.

By setting the data mapping method attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's data mapping method entry to define the method for mapping color, or other application-specific data, to intrinsic color. This is the first stage in the color pipeline. When you set the data mapping method attribute's ASF to **PEXBundled**, the implementation uses the data mapping method from the data mapping table.

If you do not set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**), then the implementation applies this data mapping method to back-facing as well as front-facing surfaces of output primitives**.** Otherwise the implementation uses the back-facing data mapping method (see **PEXOCCSetBFDataMappingMethod**).

PEXlib defines the following methods (*method*):

> **PEXDataMapNone** - prioritizes the source of the intrinsic color, from first to last, as vertex color (if defined with the primitive), facet color (if defined with the primitive), and surface color. This method does not require any data.**(5.2)**
>
> **PEXDataMapColor** - defines a prioritized list, which includes at least one of three sources. The source of intrinsic color is the vertex color (if defined with the primitive), facet color (if defined with the primitive), or surface color. The prioritized list starts with your top choice. If there is no match between the prioritized list of source selectors and the data defined with a primitive being data mapped, or if the source selectors are otherwise invalid, then the surface color is used as the source.**(5.2)**

If the implementation does not support the specified data mapping *method*, then the method defaults to **PEXDataMapNone**.

If the data record size (*data_rec_len*) does not match the data mapping method (*method*) or if you specify an undefined source selector, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the data mapping attribute set.

Data Structures

The method **PEXDataMapNone** requires no data record.

The method **PEXDataMapColor** requires a list of source selectors chosen from the set (**PEXDMMSurfaceColor, PEXDMMVertexColor, PEXDMMFacetColor**).

Errors

BadPEXOutputCommand	The data record is incorrect.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCSetBFDataMappingMethod, PEXOCCSetIndividualASF

PEXOCCSetDepthCueIndex - Set Depth Cue Index (5.0)

Synopsis

void **PEXOCCSetDepthCueIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The depth cue table index.

Returns

None.

Description

Invoke **PEXOCCSetDepthCueIndex** to create an attribute output command which sets the depth-cue index attribute in the renderer's pipeline state according to the value specified by *index*.

Use the depth cue index to select an entry of the renderer's depth cue table. Depth cueing blends the color computed by the lighting, shading, and texturing stages of the rendering pipeline with a depth cue color. The blending is controlled relative to the depth in normalized projection coordinate (NPC) space. Use depth cueing, for example, to blend with the background color the portion of an image that is furthest away from the viewer. This heightens the sensation of depth in the image.

If you specify an undefined depth cue *index*, then the index defaults to zero. If index zero is defined, then depth cueing is turned off. If you specify an *index* which exceeds 65534, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the global attribute set and affects all output primitives.

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

PEXOCCSetDrawingFunction - Set Drawing Function (5.2)

Synopsis

void **PEXOCCSetDrawingFunction**(`PEXOCC` *context*, unsigned int *function*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
function	The drawing function.

Returns

None.

Description

Invoke **PEXOCCSetDrawingFunction** to set the drawing function attribute of the renderer's pipeline state according to the value specified by *function*.

Define the drawing function to specify how the implementation, after color approximation, combines newly drawn pixels with existing pixels in the drawable. This function applies only to image planes and to the rendering of all output primitives.

The registered values, derived from Xlib definitions (defined in *X.h*), for the drawing functions are shown in Table 9:

Table 9: Drawing Functions

Symbol	Description
GXclear	all 0's
GXand	src AND dest
GXandReverse	src AND (NOT dest)
GXcopy	src
GXandInverted	(NOT src) AND dest
GXnoop	dst
GXxor	src XOR dest
GXor	src OR dest
GXnor	(NOT src) AND (NOT dest)
GXequiv	(NOT src) XOR dest
GXinvert	NOT dest
GXorReverse	src OR (NOT dest)
GXcopyInverted	NOT src
GXorInverted	(NOT src) OR dest

Table 9: Drawing Functions

Symbol	Description
GXnand	(NOT src) OR (NOT dest)
GXset	all 1's

If you enabled echoing with method **PEXEchoXORColor**, then the drawing function you select is overridden. The drawing functions, except *GXcopy*, interact with alpha blending, antialiasing, transparency, and plane masking, except for all planes enabled, in an implementation-dependent way.

If you specify an undefined drawing *function* value, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the global attribute set and affects all output primitives.

Errors

BadPEXOutputCommand	The specified function is not registered.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCSetEdgeBundleIndex - Set Edge Bundle Index (5.0)

Synopsis

void **PEXOCCSetEdgeBundleIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The edge bundle table index.

Returns

None.

Description

Invoke **PEXOCCSetEdgeBundleIndex** to create an attribute output command which sets the edge bundle index attribute in the renderer's pipeline state according to the value specified by *index*.

You use the edge bundle index to select an entry of the renderer's edge bundle table whenever you have set a bundlable edge attribute's ASF to **PEXBundled** (see **PEXOCCSetIndividualASF**). PEXlib supports the bundling of the following edge attributes: surface edge flag, surface edge type, surface edge width, and surface edge color.

If you specify an undefined edge bundle *index*, then the index defaults to edge bundle index one. If index one is undefined, then the implementation sets the bundled surface edge flag to **PEXOff**, the surface edge type to **PEXSurfaceEdgeSolid**, the surface edge width to 1.0, and the surface edge color to index color one. If you specify an edge bundle *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the general edge attribute set.

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

PEXOCCSetIndividualASF, **PEXCreateLookupTable**

PEXOCCSetFacetCullingMode - Set Facet Culling Mode (5.0)

Synopsis

void **PEXOCCSetFacetCullingMode**(`PEXOCC` *context*, int *mode*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
mode	The facet culling mode (**PEXNone**, **PEXBackFaces** or **PEXFrontFaces**).

Returns

None.

Description

Invoke **PEXOCCSetFacetCullingMode** to create an attribute output command which sets the facet culling mode attribute in the renderer’s pipeline state.

PEXlib defines the following facet culling modes (*mode*):

PEXNone - does not render any facets.**(5.0)**

PEXBackFaces - renders front-facing facets and does not render back-facing facets. **(5.0)**

PEXFrontFaces - renders back-facing facets and does not render front-facing facets.**(5.0)**

A front-facing facet is defined if the geometric normal of a primitive facet has a non-negative z-component in normalized projection coordinates (NPC’s).

If you specify an undefined facet culling *mode*, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the general surface attribute set.

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

PEXOCCSetFacetDistinguishFlag - Set Facet Distinguish Flag (5.0)

Synopsis

void **PEXOCCSetFacetDistinguishFlag**(`PEXOCC` *context*, int *flag*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
flag	The facet distinguish flag (**True** or **False**).

Returns

None.

Description

Invoke **PEXOCCSetFacetDistinguishFlag** to create an attribute output command which sets the facet distinguish flag attribute in the renderer's pipeline state according to the value specified by *flag*.

If you specify the facet distinguish flag as **True**, then the implementation applies back-facing surface attributes to back-facing facets, over-riding the corresponding surface attributes. If **False**, then the implementation applies surface attributes to both front-facing and back-facing facets.

A surface primitive's facet is back-facing if its geometric normal has a negative z-component in normalized projection coordinates (NPC's).

If you do not specify **True** or **False** for the facet distinguish *flag*, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the general surface attribute set.

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

PEXOCCSetGlobalTransform - Set Global Transformation 3D (5.0)

Synopsis

void **PEXOCCSetGlobalTransform**(`PEXOCC` *context*, `PEXMatrix` *transform*, `PEXMatrixHints` *hints*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
transform	The new global transformation matrix [§2.3.10, page 28].
hints	Hints for the new global transformation matrix [§2.3.11, page 28] (**PEXMHNone**, **PEXMHIdentity**, **PEXMHTranslation**, **PEXMHRotation**, **PEXMHReflection**, **PEXMHUniformScale**, **PEXMHNonUniformScale**, **PEXMHShear**, **PEXMHPerspProjective**, **PEXMHNonPerspProjective**, **PEXMH3DTo2DParProjection**, **PEXMH3DTo2DPerspProjection**, **PEXMHZero**, **PEXMHSingularUnknown**, **PEXMHNonSingularUnknown**). Available only in the OCC form of the function.

Returns

None.

Description

Invoke **PEXOCCSetGlobalTransform** to create an attribute output command which replaces the current global transformation matrix with the specified matrix in the renderer's pipeline state.

The **PEXOCCSetGlobalTransform** function generates a **PEXOCGlobalTransformWithHints** output command. However, if the PEX server on the *display* connection is version 5.1 or the Alternate Protocol Level is 5.1, then this function generates a **PEXOCGlobalTransform** output command, ignoring any supplied hint.

The **PEXSetGlobalTransform** function generates a **PEXOCGlobalTransform** output command.

This function stores the matrix in row-major order.

PEXlib defines the following matrix hints (*hints*):(5.2)

PEXMHNone - indicates that there is no additional information about the contents of the matrix.

PEXMHIdentity - indicates that the matrix contains:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

PEXMHTranslation - indicates that the matrix is non-singular, affine, angle-preserving, Euclidean and of one of the following forms, or a product of these forms. The fourth column in the fourth row is one.

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

PEXMHRotation - indicates that the matrix is non-singular, affine, angle-preserving, Euclidean and of one of the following forms, or a product of these forms. The fourth column in the fourth row is one.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\Theta_x & -\sin\Theta_x & 0 \\ 0 & \sin\Theta_x & \cos\Theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\Theta_y & 0 & \sin\Theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\Theta_y & 0 & \cos\Theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\Theta_z & -\sin\Theta_z & 0 & 0 \\ \sin\Theta_z & \cos\Theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

PEXMHReflection - indicates that the matrix is a non-singular, affine, angle-preserving, Euclidean transform with a negative determinant. The fourth column in the fourth row is one.

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

PEXMHUniformScale - indicates that the matrix is non-singular, affine, angle-preserving. The fourth column in the fourth row is one.

$$\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

PEXMHNonUniformScale - indicates that the matrix is non-singular, affine and of one of the following forms, or a product of these forms. The fourth column in the fourth row is one.

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

PEXMHShear - indicates that the matrix is non-singular, affine and of one of the following forms, or a product of these forms. The fourth column in the fourth row is one.

$$\begin{bmatrix} 1 & S_{xy} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ S_{yx} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & S_{xz} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ S_{zx} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & S_{yz} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & S_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

PEXMHPerspProjective - indicates that the matrix is 3D to 3D projective, non-singular, non-affine and of one of the following forms, or a product of these forms. All columns and rows have at least one non-zero value. One of the first three columns in the fourth row is non-zero (r != 0). The fourth column of the fourth row is one.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & r & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & r & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ r & 0 & 0 & 1 \end{bmatrix}$$

PEXMHNonPerspProjective - indicates that the matrix is 3D to 3D projective, non-singular, non-affine. All columns and rows have at least one non-zero value. The fourth column of the fourth row is not equal to zero or one.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix}$$

PEXMH3DTo2DParProjection - indicates that the matrix is singular, affine. One of the first three rows is all zeros. All of the first three columns in the fourth row are zero. The fourth column in the fourth row is one.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

PEXMH3DTo2DPerspProjection - indicates that the matrix is singular, non-affine. One of the first three rows is all zeros. The corresponding column of the fourth row is non-zero (r != 0). The fourth column in the fourth row is one.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & r & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & r & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ r & 0 & 0 & 1 \end{bmatrix}$$

PEXMHZero - indicates that the matrix is composed of all zero entries:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

PEXMHSingularUnknown - indicates that the matrix has a zero determinant. No other information is available.

PEXMHNonSingularUnknown - indicates that the matrix has a non-zero determinant. No other information is available.

You may supply more than one matrix hint by or’ing them together. For example, if you form a composite matrix transform by multiplying a translation matrix and a rotation matrix together, or by simply forming one manually or with a utility, the hint is the logical OR of **PEXMHTranslation** and **PEXMHRotation**.

If you provide undefined hints or hints which do not accurately describe the matrix, then the result is implementation-dependent. If you do not provide any matrix hints by using the **PEXSetGlobalTransform** (non-OCC) function, then the implementation assumes the hint is **PEXMHNone**. The non-OCC version of the interface does not support the hints argument, for backwards compatibility with previous versions of PEXlib.

The implementation is not required to utilize the hints and may therefore set the pipeline state's global transform hint to **PEXMHNone**.

This function is a member of the global attribute set.

Errors

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

See Also

PEXOCCSetGlobalTransform2D

PEXOCCSetGlobalTransform2D - Set Global Transformation 2D (5.0)

Synopsis

void **PEXOCCSetGlobalTransform2D**(`PEXOCC` *context*, `PEXMatrix3x3` *transform*, `PEXMatrixHints` *hints*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
transform	The new global transformation matrix [§2.3.9, page 28].
hints	Hints for the new global transformation matrix [§2.3.11, page 28] (**PEXMHNone**, **PEXMHIdentity**, **PEXMHTranslation**, **PEXMHRotation**, **PEXMHReflection**, **PEXMHUniformScale**, **PEXMHNonUniformScale**, **PEXMHShear**, **PEXMHPerspProjective**, **PEXMHNonPerspProjective**, **PEXMH3DTo2DParProjection**, **PEXMH3DTo2DPerspProjection**, **PEXMHZero**, **PEXMHSingularUnknown**, **PEXMHNonSingularUnknown**). Available only in the OCC form of the function.

Returns

None.

Description

Invoke **PEXOCCSetGlobalTransform2D** to create an attribute output command which replaces the current global transformation matrix with the specified matrix in the renderer's pipeline state.

The **PEXOCCSetGlobalTransform2D** function generates a **PEXOCGlobalTransform2DWithHints** output command. However, if the PEX server on the *display* connection is version 5.1 or the Alternate Protocol Level is 5.1, then this function generates a **PEXOCGlobalTransform2D** output command, ignoring any supplied hint.

The **PEXSetGlobalTransform2D** function generates a **PEXOCGlobalTransform2D** output command.

This attribute output command is similar to the **PEXOCCSetGlobalTransform** function except that you specify the global transformation matrix as a 3×3 matrix.

Before the implementation replaces the global transformation matrix, the 3×3 matrix is expanded to a 4 x 4 matrix as follows:

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

The matrix hints, if given, are representative of the matrix after it has been expanded.

This function is a member of the global attribute set.

Errors

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

See Also

PEXOCCSetGlobalTransform

PEXOCCSetHighlightIndex - Set Highlight Index (5.2)

Synopsis

void **PEXOCCSetHighlightIndex**(`PEXOCC` *context*, unsigned int *index*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The highlight table index.

Returns

None.

Description

Invoke **PEXOCCSetHighlightIndex** to create an attribute output command which sets the highlight table index attribute in the renderer's pipeline state according to the value specified by *index*.

Specify the highlight index to select an entry of the renderer's highlight table. When the pipeline state's *name set* and the highlight filter defined by the renderer's *highlight inclusion* and *highlight exclusion* entries indicate a primitive is to be highlighted, the selected entry of the highlight table defines the highlight method.

If you specify a highlight table *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the global attribute set.

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

PEXOCCSetHLHSRID - Set Hidden-Line Hidden-Surface Removal Identifier (5.0)

Synopsis

void **PEXOCCSetHLHSRID**(`PEXOCC` *context*, unsigned long *hlhsr_id*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
hlhsr_id	The HLHSR identifier (**PEXHLHSRIDDisable**, **PEXHLHSRIDEnable** or **PEXHLHSRIDEnableTestOnly**).

Returns

None.

Description

Invoke **PEXOCCSetHLHSRID** to create an attribute output command which sets the HLHSR identifier attribute in the renderer's pipeline state according to the value specified by *hlhsr_id*.

This output command is a no-op for all standard HLHSR modes except **PEXHLHSRZBufferID**.

PEXlib defines the following HLHSR identifiers (*hlhsr_id*) when you set the renderer's HLHSR mode to **PEXHLHSRZBufferID**:

> **PEXHLHSRIDDisable** - disables z-buffering.**(5.0)**
>
> **PEXHLHSRIDEnable** - tests each pixel's z-value against the z-buffer to determine if the pixel should be rendered. If the pixel is rendered, then the z-buffer is updated to reflect the pixel's z-value.**(5.0)**
>
> **PEXHLHSRIDEnableTestOnly** - tests each pixel's z-value against the z-buffer to determine if the pixel should be rendered. The z-buffer is *not* updated to reflect the pixel's z-value.**(5.2)**

Inquire the supported HLHSR modes by invoking the **PEXGetEnumTypeInfo** request.

If you specify an unsupported HLHSR identifier (*hlhsr_id*), then the HLHSR identifier used is implementation-dependent. For non-standard HLHSR modes, the result is implementation-dependent.

HLHSR interacts with alpha blending, antialiasing, transparency, drawing functions (except *GXcopy*), and plane masking (except all planes enabled), in an implementation-dependent way.

This function is a member of the global attribute set.

Errors

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

See Also

PEXOCCSetIndividualASF - Set Individual Aspect Source Flags (5.0)

Synopsis

void **PEXOCCSetIndividualASF**(`PEXOCC` *context*, unsigned long *attribute*, int *asf*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
attribute	The ASF attribute name (see the table below).
asf	The aspect source flag value (**PEXIndividual** or **PEXBundled**).

Returns

None.

Description

Invoke **PEXOCCSetIndividualASF** to create an attribute output command which sets the aspect source flag for a single, individual attribute in the renderer's pipeline state.

Table 10: ASF Attributes and their corresponding tables and functions

ASF *attribute* Name	Bundle Table	Individual Attribute Function	PEX
PEXASFBFDataMappingMethod	"Data Mapping Table" on page 445	"PEXOCCSetBFDataMappingMethod - Set Back-Facing Data Mapping Method (5.2)" on page 279	(5.2)
PEXASFBFInteriorStyle	"Interior Bundle Table" on page 450	"PEXOCCSetBFInteriorStyle - Set Back-Facing Surface Interior Style (5.0)" on page 283	(5.0)
PEXASFBFInteriorStyleIndex	"Interior Bundle Table" on page 450	"PEXOCCSetBFInteriorStyleIndex - Set Back-Facing Surface Interior Style Index (5.0)" on page 285	(5.0)
PEXASFBFReflectionAttr	"Interior Bundle Table" on page 450	"PEXOCCSetBFReflectionAttributes - Set Back-Facing Surface Reflection Attributes (5.0)" on page 288	(5.0)

Table 10: ASF Attributes and their corresponding tables and functions

ASF *attribute* Name	Bundle Table	Individual Attribute Function	PEX
PEXASFBFReflectionModel	"Interior Bundle Table" on page 450	"PEXOCCSetBFReflectionModel - Set Back-Facing Surface Reflection Model (5.0)" on page 290	(5.0)
PEXASFBFSurfaceColor	"Interior Bundle Table" on page 450	"PEXOCCSetBFSurfaceColor - Set Back-Facing Surface Color (5.0)" on page 294[a]	(5.0)
PEXASFBFSurfaceInterp	"Interior Bundle Table" on page 450	"PEXOCCSetBFSurfaceInterpMethod - Set Back-Facing Surface Interpolation Method (5.0)" on page 297	(5.0)
PEXASFCharExpansion	"Text Bundle Table" on page 458	"PEXOCCSetCharExpansion - Set Character Expansion Factor (5.0)" on page 299	(5.0)
PEXASFCharSpacing	"Text Bundle Table" on page 458	"PEXOCCSetCharSpacing - Set Character Spacing (5.0)" on page 302	(5.0)
PEXASFCurveApprox	"Line Bundle Table" on page 454	"PEXOCCSetCurveApprox - Set Curve Approximation Method (5.0)" on page 305	(5.0)
PEXASFDataMappingMethod	"Data Mapping Table" on page 445	"PEXOCCSetDataMappingMethod - Set Data Mapping Method (5.2)" on page 308	(5.2)
PEXASFInteriorStyle	"Interior Bundle Table" on page 450	"PEXOCCSetInteriorStyle - Set Surface Interior Style (5.0)" on page 331	(5.0)
PEXASFInteriorStyleIndex	"Interior Bundle Table" on page 450	"PEXOCCSetInteriorStyleIndex - Set Surface Interior Style Index (5.0)" on page 333	(5.0)
PEXASFLineCapStyle	"Line Bundle Table" on page 454	"PEXOCCSetLineCapStyle - Set Line Cap Style (5.2)" on page 338	(5.2)
PEXASFLineColor	"Line Bundle Table" on page 454	"PEXOCCSetLineColor - Set Line Color (5.0)" on page 339[a]	(5.0)
PEXASFLineJoinStyle	"Line Bundle Table" on page 454	"PEXOCCSetLineJoinStyle - Set Line Join Style (5.2)" on page 342	(5.2)
PEXASFLineType	"Line Bundle Table" on page 454	"PEXOCCSetLineType - Set Line Type (5.0)" on page 345	(5.0)

Table 10: ASF Attributes and their corresponding tables and functions

ASF *attribute* Name	Bundle Table	Individual Attribute Function	PEX
PEXASFLineWidth	"Line Bundle Table" on page 454	"PEXOCCSetLineWidth - Set Line Width (5.0)" on page 347	(5.0)
PEXASFMarkerColor	"Marker Bundle Table" on page 455	"PEXOCCSetMarkerColor - Set Marker Color (5.0)" on page 353[a]	(5.0)
PEXASFMarkerScale	"Marker Bundle Table" on page 455	"PEXOCCSetMarkerScale - Set Marker Scale (5.0)" on page 356	(5.0)
PEXASFMarkerType	"Marker Bundle Table" on page 455	"PEXOCCSetMarkerType - Set Marker Type (5.0)" on page 357	(5.0)
PEXASFParaSurfCharacteristics	"Interior Bundle Table" on page 450	"PEXOCCSetParaSurfCharacteristics - Set Parametric Surface Characteristics (5.0)" on page 364	(5.2)
PEXASFPolylineInterp	"Line Bundle Table" on page 454	"PEXOCCSetPolylineInterpMethod - Set Polyline Interpolation Method (5.0)" on page 374	(5.0)
PEXASFReflectionAttr	"Interior Bundle Table" on page 450	"PEXOCCSetReflectionAttributes - Set Surface Reflection Attributes (5.0)" on page 379	(5.0)
PEXASFReflectionModel	"Interior Bundle Table" on page 450	"PEXOCCSetReflectionModel - Set Surface Reflection Model (5.0)" on page 381	(5.0)
PEXASFSurfaceApprox	"Interior Bundle Table" on page 450	"PEXOCCSetSurfaceApprox - Set Surface Approximation Method (5.0)" on page 387	(5.0)
PEXASFSurfaceColor	"Interior Bundle Table" on page 450	"PEXOCCSetSurfaceColor - Set Surface Color (5.0)" on page 390[a]	(5.0)
PEXASFSurfaceEdgeColor	"Edge Bundle Table" on page 447	"PEXOCCSetSurfaceEdgeColor - Set Surface Edge Color (5.0)" on page 393	(5.0)
PEXASFSurfaceEdgeType	"Edge Bundle Table" on page 447	"PEXOCCSetSurfaceEdgeType - Set Surface Edge Type (5.0)" on page 396	(5.0)
PEXASFSurfaceEdgeWidth	"Edge Bundle Table" on page 447	"PEXOCCSetSurfaceEdgeWidth - Set Surface Edge Width (5.0)" on page 398	(5.0)
PEXASFSurfaceEdges	"Edge Bundle Table" on page 447	"PEXOCCSetSurfaceEdgeFlag - Set Surface Edge Flag (5.0)" on page 395	(5.0)

Table 10: ASF Attributes and their corresponding tables and functions

ASF *attribute* Name	Bundle Table	Individual Attribute Function	PEX
PEXASFSurfaceInterp	"Interior Bundle Table" on page 450	"PEXOCCSetSurfaceInterpMethod - Set Surface Interpolation Method (5.0)" on page 399	(5.0)
PEXASFTextColor	"Text Bundle Table" on page 458	"PEXOCCSetTextColor - Set Text Color (5.0)" on page 404[a]	(5.0)
PEXASFTextFontIndex	"Text Bundle Table" on page 458	"PEXOCCSetTextFontIndex - Set Text Font Index (5.0)" on page 407	(5.0)
PEXASFTextPrec	"Text Bundle Table" on page 458	"PEXOCCSetTextPrecision - Set Text Precision (5.0)" on page 410	(5.0)

a. This function also has a "ColorIndex" form on a subsequent page.

Use the aspect source flag (*asf* argument) to specify the source of the corresponding attribute's value. PEXlib defines the following aspect source flag values:

PEXIndividual - obtains the attribute's value directly from the pipeline state.(5.0)

PEXBundled - uses the attribute's value from its corresponding bundle table. Set the bundle table indices in the pipeline state to specify which entry of the bundle table to use. The bundle table indices include: **PEXOCCSetLineBundleIndex**, **PEXOCCSetInteriorBundleIndex**, **PEXOCCSetEdgeBundleIndex**, **PEXOCCSetMarkerBundleIndex**, **PEXOCCSetTextBundleIndex**, and additionally if the renderer's PHIGS interior bundle mode is set to **True**, **PEXOCCSetBFInteriorBundleIndex**, **PEXOCCSetReflectanceBundleIndex**, **PEXOCCSetBFReflectanceBundleIndex**, and **PEXOCCSetParaSurfBundleIndex**. (5.0)

If you do not set the aspect source flag to **PEXBundled** or **PEXIndividual,** or if you specify an undefined value for the *attribute* argument, then the implementation issues a *BadPEXOutputCommand* error.

The aspect source flags attribute is a member of the global attribute set.

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

PEXOCCSetInteriorBundleIndex - Set Interior Bundle Index (5.0)

Synopsis

void **PEXOCCSetInteriorBundleIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The interior bundle table index.

Returns

None.

Description

Invoke **PEXOCCSetInteriorBundleIndex** to create an attribute output command which sets the interior bundle index attribute in the renderer's pipeline state according to the value specified by *index*.

You use the interior bundle index to select an entry of the renderer's interior bundle table whenever a bundlable surface attribute's ASF is set to **PEXBundled** (see **PEXOCCSetIndividualASF**) and the renderer's PHIGS interior bundle mode is **False**. PEXlib supports the bundling of the following surface attributes: surface interior style, surface interior style index, surface color, surface interpolation method, reflectance attributes, back-facing surface interior style, back-facing surface interior style index, back-facing surface color, back-facing surface interpolation method, back-facing reflectance attributes, and parametric surface characteristics.

If the renderer's PHIGS interior bundle mode is **True**, then the interior bundle index still selects an entry of the interior bundle table but only when a bundlable interior attribute's ASF is set to **PEXBundled**. PEXlib supports the bundling of the following interior attributes: surface interior style, surface interior style index, surface color, and surface interpolation method. The remaining bundled surface attributes are also selected from the interior bundle table, but the back-facing interior bundle index, reflectance bundle index, back-facing reflectance bundle index, and parametric surface bundle index are used to select entries of the interior bundle table.

If you specify an undefined interior bundle *index*, then the index defaults to index one. If index one is not defined, then the implementation sets: the interior style to **PEXInteriorStyleHollow**; the interior style index to one; the surface color to indexed color one; reflection attributes to ambient, diffuse and specular coefficient to 1.0, specular concentration and transmission coefficient to 0.0 and specular color to indexed color one; reflection model to **PEXRMNone**; surface interpolation to **PEXSurfaceInterpNone**; and surface approximation to **PEXApproxImpDep** with *u* and *v* tolerances of 1.0. The back-facing attributes are set to identical values as the front-facing attributes. If you specify an interior bundle *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

See also the discussion on PHIGS interior bundle mode (page 450).

This function is a member of the general surface, styled and shaded surface, curved surface, and parametric surface attribute sets.

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, **PEXOCCSetBFInteriorBundleIndex**, **PEXOCCSetBFReflectanceBundleIndex**, **PEXOCCSetIndividualASF, PEXOCCSetParaSurfBundleIndex, PEXOCCSetReflectanceBundleIndex**

PEXOCCSetInteriorStyle - Set Surface Interior Style (5.0)

Synopsis

void **PEXOCCSetInteriorStyle**(`PEXOCC` *context*, int *style*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
style	The interior style (**PEXInteriorStyleHollow**, **PEXInteriorStyleSolid**, **PEXInteriorStylePattern**, **PEXInteriorStyleHatch**, **PEXInteriorStyleEmpty** or **PEXInteriorStyleTexture**).

Returns

None.

Description

Invoke **PEXOCCSetInteriorStyle** to create an attribute output command which sets the surface interior style attribute in the renderer's pipeline state according to the value specified by *style*.

By setting the interior style attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use pipeline state's interior style entry to describe how to fill the interior of surfaces. When you set the interior style attribute's ASF to **PEXBundled**, the implementation uses the interior style value from the interior bundle table. If you do not set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**), then the implementation applies this interior style to back-facing as well as front-facing surface interiors**.** Otherwise the implementation uses the back-facing surface interior style (see **PEXOCCSetBFInteriorStyle**).

PEXlib defines the following interior styles (*style*):

PEXInteriorStyleHollow - draws the boundary of the surface primitives using the current surface color but does not fill the interior.**(5.0)**

PEXInteriorStyleSolid - fills the interior of surface primitives using the current surface color.**(5.0)**

PEXInteriorStylePattern - fills the interior of surface primitives using the pattern defined according to the entry of the renderer's pattern table specified by the interior style index.**(5.0)**

PEXInteriorStyleHatch - fills the interior of surface primitives using the current surface color and the hatch style specified by the interior style index.**(5.0)**

PEXInteriorStyleEmpty - does not draw the interiors of the surface primitives at all.**(5.0)**

PEXInteriorStyleTexture - textures the interior of the surface primitives using the texture maps specified by the active textures.**(5.2)**

If you specify an undefined interior *style*, then the implementation issues a *BadPEXOutputCommand* error. If the implementation does not support the defined interior *style* you specify, then the style defaults to

PEXInteriorStyleHollow. Inquire the supported values for the surface interior style by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the styled and shaded surface attribute set.

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, **PEXOCCSetBFInteriorStyle, PEXOCCSetIndividualASF**

PEXOCCSetInteriorStyleIndex - Set Surface Interior Style Index (5.0)

Synopsis

void **PEXOCCSetInteriorStyleIndex**(`PEXOCC` *context*, int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The interior style index.

Returns

None.

Description

Invoke **PEXOCCSetInteriorStyleIndex** to create an attribute output command which sets the interior style index attribute in the renderer's pipeline state according to the value specified by *index*.

By setting the interior style index attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's interior style index entry to describe how the interior of surfaces is filled. When you set the interior style index attribute's ASF to **PEXBundled**, the implementation uses the interior style index from the interior bundle table. If you do not set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**), then the implementation applies this interior style index to back-facing as well as front-facing surface interiors. Otherwise the implementation uses the back-facing surface interior style index.

If you set the current interior style to **PEXInteriorStylePattern** or **PEXInteriorStyleHatch**, then the implementation uses the interior style index you specify to further define the rendering style of surface primitives. If you set the current interior style to **PEXInteriorStylePattern** without defining the specified pattern table index, then the index defaults to interior style index one.

For **PEXInteriorStyleHatch**, the interior style index determines the hatch style. PEXlib defines the following hatch styles:

> **PEXInteriorStyleHatch45Degrees** - draws the surface interior with single pixel width lines at about 45 degrees, in device coordinates.**(5.2)**
>
> **PEXInteriorStyleHatch135Degrees** - draws the surface interior with single pixel width lines at about 135 degrees, in device coordinates.**(5.2)**

If the implementation does not support the specified hatch style, then the hatch style defaults to **PEXInteriorStyleHatch45Degrees**. Inquire the supported values for the hatch style by invoking the **PEXGetEnumTypeInfo** function.

For **PEXInteriorStylePattern**, *index* specifies the index into the Pattern LUT. If the Pattern LUT entry specified by *index* is zero, the implementation uses an index of 1. For most other LUT index setting output commands, attempting to set an unsupported index, such as index 0 in a Pattern LUT, results in an Output Command error. In this case, the current pipeline state determines whether the index is used for an Interior Style index or a Pattern LUT index. Since the pipeline state is not available for an OC stored in a structure until the structure is rendered, the criteria for generating an OC error is not known until the structure is rendered. Since the PEX implementation cannot report OC errors while rendering structures, this OC therefore cannot generate OC errors; it falls back to index 1 instead.

This function is a member of the styled and shaded surface attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, PEXOCCSetBFInteriorStyle, PEXOCCSetBFInteriorStyleIndex, PEXOCCSetIndividualASF

PEXOCCSetLightSourceState - Set Light Source State (5.0)

Synopsis

void **PEXOCCSetLightSourceState**(`PEXOCC` *context*, unsigned int *enable_count*, `PEXTableIndex` **enable*, unsigned int *disable_count*, `PEXTableIndex` **disable*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	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 [§2.3.21, page 29].
disable_count	The number of lights to disable.
disable	An array of lights to disable [§2.3.21, page 29].

Returns

None.

Description

Invoke **PEXOCCSetLightSourceState** to create an attribute output command which modifies the light source state attribute in the renderer's pipeline state.

If the pipeline's state surface reflection model or back-facing surface reflection model is any value other than **PEXRMNone**, then lighting is enabled. The light source state maintains a list of indices into the renderer's light table, each of which defines a light, to define which lights are on.

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.

If you list a light in both the enable list (*enable*) and the disable list (*disable*), or specify a light index of zero, then the implementation issues a *BadPEXOutputCommand* error. If any light in the enable or disable list references an undefined light table entry, then the implementation ignores the light. If you enable more non-ambient lights than the maximum number supported by the implementation in reflectance calculations, then which lights are used is implementation-dependent. Inquire the maximum number of non-ambient lights by invoking the **PEXGetImpDepConstants** function.

This function is a member of the general surface attribute set.

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, PEXGetImpDepConstants, **PEXOCCSetBFReflectionModel**, **PEXOCCSetReflectionModel**

PEXOCCSetLineBundleIndex - Set Line Bundle Index (5.0)

Synopsis

void **PEXOCCSetLineBundleIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The line bundle table index.

Returns

None.

Description

Invoke **PEXOCCSetLineBundleIndex** to create an attribute output command which sets the line bundle index attribute in the renderer's pipeline state according to the value specified by *index*.

You use the line bundle index to select an entry of the renderer's line bundle table whenever a bundlable line attribute's ASF is set to **PEXBundled** (see **PEXOCCSetIndividualASF**). PEXlib supports the bundling of the following line attributes: line type, polyline interpolation, curve approximation, line width, line cap style, line join style and line color.

If you specify an undefined line bundle *index*, then the index defaults to one. If index one is not defined, then the implementation sets the bundled line type to **PEXLineTypeSolid**; polyline interpolation to **PEXPolylineInterpNone**; curve approximation to **PEXApproxImpDep** with a tolerance of 1.0; line width to 1.0, line cap style to **PEXLineCapStyleButt**, line join style to **PEXLineJoinStyleImpDep**, and line color to index color one. If you specify a line bundle *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the line, shaded line, and curve attribute set.

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, **PEXOCCSetIndividualASF**

PEXOCCSetLineCapStyle - Set Line Cap Style (5.2)

Synopsis

void **PEXOCCSetLineCapStyle**(PEXOCC *context*, unsigned long *cap_style*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
cap_style	The desired cap style (**PEXLineCapButt**, **PEXLineCapRound**, or **PEXLineCapProject**).

Returns

None.

Description

Invoke **PEXOCCSetLineCapStyle** to set the line cap style entry of the renderer's pipeline state according to the value specified by *cap_style*.

By setting the line cap style attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's line cap style entry to provide control over how the implementation renders the ends of a wide line.When you set the line cap style attribute's ASF to **PEXBundled**, the implementation uses the line cap style from the line bundle table.

PEXlib defines the following cap styles (*cap_style*)

PEXLineCapButt - The wide line is square at the endpoint (perpendicular to the slope of the line) with no projection beyond. **(5.2)**

PEXLineCapRound - The wide line has a circular arc with the diameter equal to the linewidth, centered on the endpoint. **(5.2)**

PEXLineCapProject - The wide line is square at the end, but the path continues beyond the endpoint for a distance equal to half the linewidth. **(5.2)**

If the implementation does not support the specified line cap style (*cap_style*), then the cap style defaults to **PEXLineCapButt**. Inquire the supported values for the line *cap style* by invoking the **PEXGetEnumTypeInfo** function

This function is a member of the line attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, PEXOCCSetIndividualASF

PEXOCCSetLineColor - Set Line Color (5.0)

Synopsis

void **PEXOCCSetLineColor**(`PEXOCC` *context*, `PEXColor` **color*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
color_type (OCC)	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
color	A pointer to the line color [§2.5.12, page 32].

Returns

None.

Description

Invoke **PEXOCCSetLineColor** to create an attribute output command which sets the line color attribute in the renderer's pipeline state according to the value specified by *color*.

By setting the line color attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's line color entry to select the color of a specified line. When you set the line color attribute's ASF to **PEXBundled**, the implementation uses the line color from the line bundle table.

Depending on the *color type* you specify, this function sets the attribute to either an indexed color or a direct color value. If you specify a *color type* of **PEXColorTypeIndexed**, but you specified an undefined index, then the index defaults to color index one. If color index one is not defined, then the line color defaults to white. If you specified a color index greater than 65534, then the implementation issues a *BadPEXOutputCommand* error. If the *color type* is one of the supported direct color types and you specify an out of range component, then the result is implementation-dependent. If the *color type* is not supported, then indexed color one is used. Inquire the supported values for color type by invoking the **PEXGetEnumTypeInfo** function.

The line color function is a member of the line attribute set.

Errors

BadPEXOutputCommand	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

PEXCreateLookupTable, **PEXGetEnumTypeInfo, PEXOCCSetIndividualASF**

PEXOCCSetLineColorIndex - Set Line Color Index (5.0)

Synopsis

void **PEXOCCSetLineColorIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The color table index for lines.

Returns

None.

Description

Invoke **PEXOCCSetLineColorIndex** to set the line color attribute to an indexed color in the renderer's pipeline state according to the value specified by *index*. The index selects an entry of the renderer's color table. You can alternately use the **PEXOCCSetLineColor** function which allows you to specify a color of any type.

By setting the line color index attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's line color entry to determine the color of line attributes. When you set the line color attribute's ASF to **PEXBundled**, the implementation uses the line color from the line bundle table.

If you specify an undefined *index*, then the index defaults to color index one. If color index one is not defined, then the color defaults to white. If you specify a color *index* which is greater than 65534, then the implementation issues a *BadPEXOutputCommand* error.

The line color index function is a member of the line attribute set.

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

PEXCreateLookupTable, **PEXOCCSetIndividualASF**

PEXOCCSetLineJoinStyle - Set Line Join Style (5.2)

Synopsis

void **PEXOCCSetLineJoinStyle**(`PEXOCC` *context*, unsigned long *join_style*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
join_style	The desired join style (see Description).

Returns

None.

Description

Invoke **PEXOCCSetLineJoinStyle** to create an output primitive which sets the line join style attribute in the renderer's pipeline state according to the value specified by *join_style*.

By setting the line join style index attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's join style entry to control the rendering of the intersection between adjacent segments of a wide line. When you set the line join style attribute's ASF to **PEXBundled**, the implementation uses the join style from the line bundle table.

PEXlib defines the following line join styles (*join_style*):

PEXLineJoinImpDep - joins the wide line segments in an implementation-dependent way. **(5.2)**

PEXLineJoinRound - joins two wide line segments by rendering a circular arc with a diameter equal to the linewidth and with a centerpoint that is the vertex between the two segments. **(5.2)**

PEXLineJoinMiter - extends the outside edges of the wide line segments until they meet at an angle. However if the line miter limit (see **PEXOCCSetLineMiterLimit**) is exceeded, style **PEXLineJoin-Miter** is rendered as style **PEXLineJoinBevel**. **(5.2)**

PEXLineJoinBevel - joins the two wide line segments by applying cap style **PEXLineCapButt** (see **PEXOCCSetLineCapStyle**) to each of them and rendering the triangular notch left between them. **(5.2)**

If the implementation does not support the specified line join style (*join_style*), then the join style defaults to **PEXLineJoinImpDep**. Inquire the supported values for line *join style* by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the line attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, PEXOCCSetIndividualASF, PEXOCCSetLineCapStyle, PEXOCCSetLineMiterLimit

PEXOCCSetLineMiterLimit - Set Line Miter Limit (5.2)

Synopsis

void **PEXOCCSetLineMiterLimit**(`PEXOCC` *context*, double *miter_limit*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
miter_limit	The desired miter limit.

Returns

None.

Description

Invoke **PEXOCCSetLineMiterLimit** to set the line miter limit entry of the renderer’s pipeline state according to the value specified by *miter_limit*.

The line *miter_limit* determines the treatment of wide line segments joined by line join style **PEXLineJoinMiter** (see **PEXOCCSetLineJoinStyle**). If the ratio of the miter length (the distance between the inside and outside intersections of the two segments joined) to the line width exceeds the line miter limit, then the implementation interprets the line join style as **PEXLineJoinBevel**.

If you specify a *miter limit* that is less than 1.0, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the line attribute set.

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

PEXOCCSetLineJoinStyle

PEXOCCSetLineType - Set Line Type (5.0)

Synopsis

void **PEXOCCSetLineType**(`PEXOCC` *context*, int *line_type*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
line_type	The line type (**PEXLineTypeSolid**, **PEXLineTypeDashed**, **PEXLineTypeDotted**, **PEXLineTypeDashDot**, **PEXLineTypeCenterLine** or **PEXLineTypePhantom**).

Returns

None.

Description

Invoke **PEXOCCSetLineType** to create an attribute output command which sets the line type attribute in the renderer's pipeline state according to the value specified by *line_type*.

By setting the line type attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's line type entry to define the line types used when rendering line primitives. When you set the line type attribute's ASF to **PEXBundled**, the implementation uses the line type from the line bundle table.

PEXlib defines the following line types (*line_type*):

> **PEXLineTypeSolid** - draws the polyline or curve with a solid, unbroken line.**(5.0)**
>
> **PEXLineTypeDashed** - draws the polyline or curve with a line that is broken into even segments.**(5.0)**
>
> **PEXLineTypeDotted** - draws the polyline or curve with a line that is broken into dots.**(5.0)**
>
> **PEXLineTypeDashDot** - draws the polyline or curve with a line that contains alternating dotted and dashed line segments.**(5.0)**
>
> **PEXLineTypeCenterLine** - draws the polyline or curve with a line that contains alternating short line segments (dashes) and long dashes.**(5.2)**
>
> **PEXLineTypePhantom** - draws the polyline or curve with a line that contains a sequence of a long line segment (dash) followed by two short dashes.**(5.2)**

Whether line types, except **PEXLineTypeSolid**, are restarted or continued at the start of the polyline, at the start of a clipped segment of a polyline, or at each vertex of a polyline is implementation-dependent.

If the implementation does not support the specified *line* type (*line_type*), then the line type defaults to **PEXLineTypeSolid**. Inquire the supported values for the *line type* by invoking the **PEXGetEnumTypeInfo** function.

The line type function is a member of the line attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, **PEXOCCSetIndividualASF**

PEXOCCSetLineWidth - Set Line Width (5.0)

Synopsis

void **PEXOCCSetLineWidth**(`PEXOCC` *context*, double *width*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
width	The line width scale factor.

Returns

None.

Description

Invoke **PEXOCCSetLineWidth** to create an attribute output command which sets the line width attribute in the renderer's pipeline state according to the value specified by *width*.

By setting the line width attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's line width entry to set the width size used when rendering lines. When you set the line width attribute's ASF to **PEXBundled**, the implementation uses the line width from the line bundle table.

Use *width* as a scale factor to increase or decrease the width (in pixels) from the nominal line width for the display device. PEXlib maps the results to the nearest supported line width. Specifying a negative line width results in the thinnest line supported.

Inquire the nominal line width, the minimum line width, the maximum line width, and the number of supported line widths by invoking the **PEXGetImpDepConstants** function.

The line width is a member of the line attribute set.

Errors

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

See Also

PEXGetImpDepConstants, **PEXOCCSetIndividualASF**

PEXOCCSetLocalTransform - Set Local Transformation 3D (5.0)

Synopsis

void **PEXOCCSetLocalTransform**(`PEXOCC` *context*, int *composition*, `PEXMatrix` *transform*, `PEXMatrixHints` *hints*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	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** or **PEXReplace**).
transform	The new local transformation matrix [§2.3.10, page 28].
hints	Hints for the new local transformation matrix [§2.3.11, page 28] (**PEXMHNone**, **PEXMHIdentity**, **PEXMHTranslation**, **PEXMHRotation**, **PEXMHReflection**, **PEXMHUniformScale**, **PEXMHNonUniformScale**, **PEXMHShear**, **PEXMHPerspProjective**, **PEXMHNonPerspProjective**, **PEXMH3DTo2DParProjection**, **PEXMH3DTo2DPerspProjection**, **PEXMHZero**, **PEXMHSingularUnknown**, **PEXMHNonSingularUnknown**). Available only in the OCC form of the function.

Returns

None.

Description

Invoke **PEXOCCSetLocalTransform** to create an attribute output command which modifies the local transformation matrix in the renderer's pipeline state.

The **PEXOCCSetLocalTransform** function generates a **PEXOCLocalTransformWithHints** output command. However, if the PEX server on the *display* connection is version 5.1 or the Alternate Protocol Level is 5.1, then this function generates a **PEXOCLocalTransform** output command, ignoring any supplied hint.

The **PEXSetLocalTransform** function generates a **PEXOCLocalTransform** output command.

PEXlib defines the following composition types (*composition*):(5.0)

> **PEXPreConcatenate** - pre-concatenates the matrix you specified to the current local transformation matrix: (L' = L × T).
>
> **PEXPostConcatenate** - post-concatenates the matrix you specified to the current local transformation matrix: (L' = T × L).
>
> **PEXReplace** - replaces the current local transformation matrix with the matrix you specified: (L' = T).

See "PEXOCCSetGlobalTransform - Set Global Transformation 3D (5.0)" on page 316 for the definition of the matrix hints (*hints*):(5.2).

You may supply more than one matrix hint by or'ing them together.

If you provide undefined hints or hints which do not accurately describe the matrix, then the result is implementation-dependent.

If you use the **PEXSetLocalTransform** function, then the implementation assumes the hint is **PEXMHNone**. The non-OCC version of the interface does not support the hints argument, for backwards compatibility with previous versions of PEXlib.

The implementation is not required to use the hints and may therefore set the pipeline state's local transform hints to **PEXMHNone**.

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

If the composition type is not valid, then the implementation issues a *BadPEXOutputCommand* error.

The local transform is a member of the global attribute set.

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

PEXOCCSetLocalTransform2D

PEXOCCSetLocalTransform2D - Set Local Transformation 2D (5.0)

Synopsis

void **PEXOCCSetLocalTransform2D**(`PEXOCC` *context*, int *composition*, `PEXMatrix3x3` *transform*, `PEXMatrixHints` *hints*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	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** or **PEXReplace**).
transform	The new local transformation matrix [§2.3.9, page 28].
hints	Hints for the new global transformation matrix [§2.3.11, page 28] (**PEXMHNone**, **PEXMHIdentity**, **PEXMHTranslation**, **PEXMHRotation**, **PEXMHReflection**, **PEXMHUniformScale**, **PEXMHNonUniformScale**, **PEXMHShear**, **PEXMHPerspProjective**, **PEXMHNonPerspProjective**, **PEXMH3DTo2DParProj**, **PEXMH3DTo2DPerspProj**, **PEXMHZero**, **PEXMHSingularUnknown**, **PEXMHNonSingularUnknown**). Available only in the OCC form of the function.

Returns

None.

Description

Invoke **PEXOCCSetLocalTransform2D** to create an attribute output command which modifies the local transformation matrix in the renderer's pipeline state.

The **PEXOCCSetLocalTransform2D** function generates a **PEXOCLocalTransform2DWithHints** output command. However, if the PEX server on the *display* connection is version 5.1 or the Alternate Protocol Level is 5.1, then this function generates a **PEXOCLocalTransform2D** output command, ignoring any supplied hint.

The **PEXSetLocalTransform2D** function generates a **PEXOCLocalTransform2D** output command.

This output command is similar to the **PEXOCCSetLocalTransform** function except that you specify the local transformation matrix as a 3×3 matrix. Before modification of the local transformation matrix, the 3×3 matrix is expanded to a 4 x 4 matrix as follows:

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

The matrix hints, if given, are representative of the matrix after it has been expanded.

The local transform 2D function is a member of the global attribute set.

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

PEXOCCSetLocalTransform

PEXOCCSetMarkerBundleIndex - Set Marker Bundle Index (5.0)

Synopsis

void **PEXOCCSetMarkerBundleIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The marker bundle table index.

Returns

None.

Description

Invoke **PEXOCCSetMarkerBundleIndex** to create an attribute output command which sets the marker bundle index attribute in the renderer's pipeline state according to the value specified by *index*.

You use the marker bundle index to select an entry of the renderer's marker bundle table whenever a bundlable marker attribute's ASF is set to **PEXBundled** (see **PEXOCCSetIndividualASF**). PEXlib supports the bundling of the following marker attributes: marker type, marker scale, and marker color.

If you specify an undefined marker *index*, then the index defaults to index one. If index one is not defined, then the implementation sets the bundled marker type to **PEXMarkerAsterisk**; marker scale to 1.0 and marker color to indexed color one. If you specify a marker bundle *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the marker attribute set.

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, **PEXOCCSetIndividualASF**

PEXOCCSetMarkerColor - Set Marker Color (5.0)

Synopsis

void **PEXOCCSetMarkerColor**(`PEXOCC` *context*, `PEXColor` **color*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
color_type (OCC)	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
color	A pointer to the marker color [§2.5.12, page 32].

Returns

None.

Description

Invoke **PEXOCCSetMarkerColor** to create an attribute output command which sets the marker color attribute in the renderer's pipeline state according to the value specified by *color*.

By setting the marker color attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's marker color entry to set the rendering color for markers. When you set the marker color attribute's ASF to **PEXBundled**, the implementation uses the marker color from the marker bundle table.

Depending on the *color type* you specify, this function sets the attribute to either an indexed color or a direct color value. If you specify a *color type* of **PEXColorTypeIndexed**, but you specified an undefined index, then the index defaults to color index one. If color index one is not defined, then the marker color defaults to white. If you specified a color index greater than 65534, then the implementation issues a *BadPEXOutputCommand* error. If the *color type* is one of the supported direct color types and you specify an out of range component, then the result is implementation-dependent. If the *color type* is not supported, then indexed color one is used. Inquire the supported values for color type by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the marker attribute set.

Errors

BadPEXOutputCommand	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

PEXCreateLookupTable, PEXGetEnumTypeInfo, PEXOCCSetIndividualASF

PEXOCCSetMarkerColorIndex - Set Marker Color Index (5.0)

Synopsis

void **PEXOCCSetMarkerColorIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The color table index for markers.

Returns

None.

Description

Invoke **PEXOCCSetMarkerColorIndex** to set the marker color entry of the renderer's pipeline state to an indexed color according to the value specified by *index*. The index selects an entry of the renderer's color table. You can alternately use the **PEXOCCSetMarkerColor** function which allows you to specify a color of any type.

By setting the marker color index attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's marker color entry to determine the color of marker attributes. When you set the marker color attribute's ASF to **PEXBundled**, the implementation uses the marker color from the marker bundle table.

If you specify an undefined color *index*, then the index defaults to color index one. If color index one is not defined, then the color defaults to white. If you specify a color *index* which is greater than 65534, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the marker attribute set.

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

PEXCreateLookupTable, PEXOCCSetIndividualASF

PEXOCCSetMarkerScale - Set Marker Scale (5.0)

Synopsis

void **PEXOCCSetMarkerScale**(`PEXOCC` *context*, double *scale*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
scale	The marker scale factor.

Returns

None.

Description

Invoke **PEXOCCSetMarkerScale** to create an attribute output command which sets the marker scale attribute in the renderer's pipeline state according to the value specified by *scale*.

By setting the marker scale attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's marker scale entry to set the size of a marker. When you set the marker scale attribute's ASF to **PEXBundled**, the implementation uses the marker scale from the marker bundle table.

PEXlib multiplies the value you specify for *scale* by the nominal marker size for the display device. Then it maps the marker size to the nearest supported marker size. A negative marker scale factor results in the smallest supported marker.

Inquire the nominal marker size, the minimum marker size, the maximum marker size, and the number of supported marker sizes by invoking the **PEXGetImpDepConstants** function.

This function is a member of the marker attribute set.

Errors

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

See Also

PEXGetImpDepConstants, **PEXOCCSetIndividualASF**

PEXOCCSetMarkerType - Set Marker Type (5.0)

Synopsis

void **PEXOCCSetMarkerType**(`PEXOCC` *context*, int *marker_type*)

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

Arguments

display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
marker_type	The marker type (**PEXMarkerDot**, **PEXMarkerCross**, **PEXMarkerAsterisk**, **PEXMarkerCircle** or **PEXMarkerX**).

Returns

None.

Description

Invoke **PEXOCCSetMarkerType** to create an attribute output command which sets the marker type attribute in the renderer's pipeline state according to the value specified by *marker_type*.

By setting the marker type attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's marker type entry to set the style for rendering markers. When you set the marker type attribute's ASF to **PEXBundled**, the implementation uses the marker type from the marker bundle table.

Use *marker_type* to specify the shape of a marker primitive. If the renderer's *marker glyph table* entry is Null, then the marker type selects one of the following marker types defined by PEXlib:

> **PEXMarkerDot** - "." (dot) displays the smallest dot supported by the implementation. The marker scale factor is ignored. The dot is positioned at the marker position.**(5.0)**
>
> **PEXMarkerCross** - "+" displays a cross or plus sign. The intersection of the cross is positioned at the marker position.**(5.0)**
>
> **PEXMarkerAsterisk** - "*" displays an asterisk or star. The intersection of the asterisk is positioned at the marker position.**(5.0)**
>
> **PEXMarkerCircle** - "o" displays a circle. The center of the circle is positioned at the marker position.**(5.0)**
>
> **PEXMarkerX** - "x" displays an x. The intersection of the x is positioned at the marker position.**(5.0)**

If the renderer's marker glyph table entry is Null and you specify a marker type (*marker_type*) that is not supported, then the marker type defaults to **PEXMarkerAsterisk**. Inquire the supported values for the *marker type* by invoking the **PEXGetEnumTypeInfo** function.

The marker type function is a member of the marker attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, **PEXOCCSetIndividualASF**

PEXOCCSetModelClipFlag - Set Model Clipping Flag (5.0)

Synopsis

void **PEXOCCSetModelClipFlag**(`PEXOCC` *context*, int *flag*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
flag	The model clipping flag (**PEXClip** or **PEXNoClip**).

Returns

None.

Description

Invoke **PEXOCCSetModelClipFlag** to create an attribute output command which sets the model clip attribute in the renderer's pipeline state according to the value specified by *flag*.

PEXlib defines two model clip flags (*flag*):

PEXClip - enables modeling clipping. **(5.0)**

PEXNoClip - disables modeling clipping.**(5.0)**

If you specify an undefined model clipping *flag*, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the global attribute set.

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

PEXOCCSetModelClipVolume, **PEXOCCSetModelClipVolume2D**, **PEXRestoreModelClipVolume**

PEXOCCSetModelClipVolume - Set Model Clip Volume 3D (5.0)

Synopsis

void **PEXOCCSetModelClipVolume**(`PEXOCC` *context*, int *op*, unsigned int *count*, `PEXHalfSpace` **half_spaces*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	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 half-spaces.
half_spaces	An array of points and normal vectors defining the model clipping volume [§2.14.1, page 39].

Returns

None.

Description

Invoke **PEXOCCSetModelClipVolume** to create an attribute output command which sets the model clip volume attribute in the renderer’s pipeline state.

Specify the model clipping volume operator (*op*) to indicate how to combine the specified list of half-spaces with the current model clipping volume to form a new model clipping volume. Primitives outside of the model clipping volume are not rendered when the pipeline’s state *model clip indicator* is **PEXClip** (see **PEXOCCSetModelClipFlag**). PEXlib defines two model clipping volume operators (*op*):

> **PEXModelClipReplace** - replaces current volume using the specified half-spaces to create a new composite modeling clipping volume.**(5.0)**

> **PEXModelClipIntersection** - intersects the specified current composite modeling clipping volume with the current volume (specified by the half-spaces).**(5.0)**

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. If the implementation does not support the specified *model clipping volume operator*, if any half-space is degenerate, or if modifying the pipeline state’s *model clipping volume* exceeds the maximum number of clipping planes the implementation can support, then the implementation does not modify the pipeline state’s current *model clipping volume*. Inquire the maximum number of model clip planes supported by invoking the **PEXGetImpDepConstants** function.

The implementation transforms each half-space by the current composite modeling transformation and combines the half-space with the current model clipping volume. The resulting model clipping volume is not affected by subsequent changes to the composite modeling transformation. If you specify no half-spaces, then these missing half-spaces define a volume which is all of modeling coordinate space, effectively performing no modeling clip.

This function is a member of the global attribute set.

Errors

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

See Also

PEXGetImpDepConstants, PEXOCCSetModelClipFlag, **PEXOCCSetModelClipVolume2D**, **PEXRestoreModelClipVolume**

PEXOCCSetModelClipVolume2D - Set Model Clip Volume 2D (5.0)

Synopsis

void **PEXOCCSetModelClipVolume2D**(`PEXOCC` *context*, int *op*, unsigned int *count*, `PEXHalfSpace2D` **half_spaces*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	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 half-spaces.
half_spaces	An array of points and normal vectors defining the model clipping volume [§2.14.2, page 39].

Returns

None.

Description

Invoke **PEXOCCSetModelClipVolume2D** to create an attribute output command which sets the model clip volume attribute in the renderer's pipeline state.

This output command is similar to the **PEXOCCSetModelClipVolume** function except that you specify the half-spaces in 2D coordinates (the z-component of each point and normal vector is always assumed to be zero).

Errors

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

See Also

PEXGetImpDepConstants, PEXOCCSetModelClipFlag, **PEXOCCSetModelClipVolume**, **PEXRestoreModelClipVolume**

PEXOCCSetParaSurfBundleIndex - Set Parametric Surface Bundle Index(5.2)

Synopsis

void **PEXOCCSetParaSurfBundleIndex**(`PEXOCC` *context*, unsigned int *index*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The parametric surface bundle table index.

Returns

None.

Description

Invoke **PEXOCCSetParaSurfBundleIndex** to set the parametric surface bundle index entry of the renderer's pipeline state according to the value specified by *index*.

You use the parametric surface bundle index to select an entry of the renderer's interior bundle table whenever a bundlable parametric surface attribute's ASF is set to **PEXBundled** (see **PEXOCCSetIndividualASF**) and the renderer's PHIGS interior bundle mode is **True**. If you set the PHIGS interior bundle mode to **False**, then the implementation uses the interior bundle index instead and ignores the parametric surface bundle index. PEXlib supports the bundling of the following parametric surface attributes: surface approximation and parametric surface characteristics.

If you specify an undefined *index*, then the index defaults to index one. If index one is not defined, then the implementation sets the bundled surface approximation to **PEXApproxImpDep** and the parametric surface characteristics to **PEXPSCNone**. If you specify a parametric surface bundle table *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the curved surface and parametric surface attribute sets.

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, **PEXOCCSetIndividualASF**

PEXOCCSetParaSurfCharacteristics - Set Parametric Surface Characteristics (5.0)

Synopsis

void **PEXOCCSetParaSurfCharacteristics**(`PEXOCC` *context*, int *psc_type*, `PEXPSCData` **characteristics*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
psc_type	The parametric surface characteristic type (**PEXPSCNone**, **PEXPSCImpDep**, **PEXPSCIsoCurves**, **PEXPSCMCLevelCurves** or **PEXPSCWCLevelCurves**).
characteristics	A pointer to data defining the parametric surface characteristics [§2.22.2, page 51].

Returns

None.

Description

Invoke **PEXOCCSetParaSurfCharacteristics** to create an attribute output command which sets the parametric surface characteristics in the renderer's pipeline state according to the value specified by *psc_type*.

By setting the parametric surface characteristics attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's parametric surface characteristics entry to apply aids when visualizing the shape of parametric surfaces. When you set the parametric surface characteristics attribute's ASF to **PEXBundled**, the implementation uses the parametric surface characteristics from the interior bundle table.

PEXlib defines the following parametric surface characteristics (*characteristics*):

PEXPSCNone - does not apply any special aids when visualizing the shape of a parametric surface. No data is required for this type.**(5.0)**

PEXPSCImpDep - applies an implementation-dependent aid when visualizing the shade of a parametric surface. Back-facing portions of the surface are not distinguished. The appearance of the representation is controlled by the appropriate set of primitive attributes for the representation. It is implementation-dependent how the representation interacts with any interior rendering indicated by the interior attributes. No data is defined for this type, therefore *characteristics* is ignored.**(5.0)**

PEXPSCIsoCurves - draws isoparametric curves on the surface. The data pointed to by *characteristics* contains the number of curves to draw in each of the parameter dimensions and their placement. If the placement is Uniform, then the specified number of curves are evenly spaced between the parameter limits of the surface; curves are also drawn at the parameter limits. If the placement is NonUniform, then the specified number of curves are evenly spaced between each pair of knots; curves are also drawn at the knots. In both cases, only the portions of the isoparametric curves are drawn that are within the interior of the surface as defined by any trimming curves.**(5.0)**

Back-facing portions of the surface are not distinguished. The tessellation and appearance of the isoparametric curves are controlled by the surface approximation criteria and the polyline attributes, respectively. The isoparametric curves are drawn in addition to any interior rendering indicated by the interior style or back interior style attributes. Isoparametric curves have higher visual priority than the primitive's filled or hollow interiors, but lower priority than the primitive's edges.

If the placement is not Uniform or NonUniform, then the implementation issues a *BadPEXOutputCommand* error.

PEXPSCMCLevelCurves - Draws level curves on the surface. The curves correspond to the intersections of the surface and a finite set of places perpendicular to a modeling coordinate direction vector. The positions of the planes are specified by a sequence of intersecting points along an infinite line defined by a modeling coordinate origin point, P_0, and a direction vector, $\vec{V}$

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

The t_i are a sequence of parameters specifying the intersection points. They are in the range:

$-\infty < t_i < \infty$

The P_i are the intersection points of the perpendicular planes with the infinite line. P_0 is a specified origin point in modeling clipping coordinates, and $\vec{V}$ is the specified direction vector in modeling coordinates. The i^{th} plane is perpendicular to the direction vector, $\vec{V}$, and intersects the infinite line at point P_i.

Back-facing portions of the surface are not distinguished. The tessellation and appearance of the level curves are controlled by the surface approximation criteria and the polyline attributes, respectively. The curves are drawn in addition to any interior rendering indicated by the interior style or back-facing interior style attributes. Level curves have higher visual priority than a primitive's filled or hollowed interiors, but lower priority than its edges.**(5.0)**

If you specify a direction vector length of zero, then the implementation issues a *BadPEXOutputCommand* error.

PEXPSCWCLevelCurves - draws level curves on the surface. This type is identical to type **PEXPSCMCLevelCurves** except that the origin point, P_0, and the direction vector, $\vec{V}$, are defined in world coordinates instead of modeling coordinates.**(5.0)**

If the implementation does not support the specified parametric surface characteristics type (*psc_type*), then the parametric surface characteristics type defaults to **PEXPSCNone**. Inquire the supported values for the *parametric surface characteristic types* by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the parametric surface attribute set.

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, PEXOCCSetIndividualASF

PEXOCCSetPatternAttributes - Set Pattern Interior Style Attributes (5.0)

Synopsis

void **PEXOCCSetPatternAttributes**(`PEXOCC` *context*, `PEXCoord` **ref_point*, `PEXVector` **vector1*, `PEXVector` **vector2*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
ref_point	The pattern reference point [§2.7.1, page 33].
vector1	The first pattern reference vector [§2.31.1, page 67].
vector2	The second pattern reference vector [§2.31.1, page 67].

Returns

None.

Description

Invoke **PEXOCCSetPatternAttributes** to create an attribute output command which sets the pattern reference point and both pattern reference vectors attributes, in modeling coordinates, in the renderer's pipeline state.

To determine if your implementation supports **PEXInteriorStylePattern**, invoke the **PEXGetEnumTypeInfo** request. If the style is supported, then the pipeline state's interior style index selects an entry of the renderer's pattern table and applies it to fill area primitives of interior style **PEXInteriorStylePattern**.

When applying interior style **PEXInteriorStylePattern** to a fill area primitive, the reference point and vectors are projected along a normal vector onto the plane of each facet of the primitive. A parallelogram is formed with is origin defined by the projected reference point. A vector (*vector1*) defines one side of the parallelogram. This vector's direction is parallel to the first projected reference vector and has a length that is equal to the absolute value of the x-component of the pattern size (see **PEXOCCSetPatternSize**). The adjacent side of the parallelogram is defined similarly by the second projected reference vector (*vector2*) and the absolute value of the y-component of the pattern size. The pattern, a list of *m* x *n* equally-sized color cells, is applied across the parallelogram such that the first cell has the parallelogram origin in one corner. The first *m* colors in the pattern correspond to the row of cells nearest to and along the first projected reference vector (*vector1*). The pattern is mapped onto the facet by replicating the pattern parallelogram in directions parallel to the parallelogram's sides until the entire facet is covered.

If you specify a surface interior style of **PEXInteriorStylePattern**, then the implementation uses the pattern reference point and the pattern reference vectors to position and scale the pattern on the surface. If you specify either of the pattern reference vectors as zero length or if the vectors are parallel, then the implementation ignores this output command and the pattern entries of the pipeline state are not updated.

This function is a member of the styled and shaded surface attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, PEXOCCSetPatternAttributes2D, PEXOCCSetPatternSize

PEXOCCSetPatternAttributes2D - Set Pattern Interior Style Reference Point (5.0)

Synopsis

void **PEXOCCSetPatternAttributes2D**(`PEXOCC` *context*, `PEXCoord2D` **ref_point*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
ref_point	The pattern reference point [§2.7.2, page 33].

Returns

None.

Description

Invoke **PEXOCCSetPatternAttributes2D** to create an attribute output command which sets the pattern reference point and both pattern reference vector attributes, in 2D modeling coordinates, in the renderer's pipeline state.

To determine if your implementation supports **PEXInteriorStylePattern**, invoke the **PEXGetEnumTypeInfo** request. If the style is supported, then the pipeline state's interior style index selects an entry of the renderer's pattern table and applies it to fill area primitives of interior style **PEXInteriorStylePattern**.

The pattern reference point is set to *ref_point* with a z coordinate of zero. The two pattern reference vectors are set to <1,0,0> and <0,1,0> (see **PEXOCCSetPatternAttributes**). If you specify the surface interior style as **PEXInteriorStylePattern**, then the implementation uses the pattern reference point and the reference vectors to position and scale the pattern on the surface.

This function is a member of the styled and shaded surface attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, PEXOCCSetPatternAttributes

PEXOCCSetPatternSize - Set Pattern Interior Style Size (5.0)

Synopsis

void **PEXOCCSetPatternSize**(`PEXOCC` *context*, double *width*, double *height*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
width	The pattern width.
height	The pattern height.

Returns

None.

Description

Invoke **PEXOCCSetPatternSize** to create an attribute output command which sets the pattern size attribute, in modeling coordinates, in the renderer's pipeline state.

This function only applies the magnitude of the specified width and height.

To determine if your implementation supports **PEXInteriorStylePattern**, invoke the **PEXGetEnumTypeInfo** request. If the style is supported, then the pipeline state's interior style index selects an entry of the renderer's pattern table and applies it to fill area primitives of interior style **PEXInteriorStylePattern**.

When applying interior style **PEXInteriorStylePattern** to a fill area primitive, the reference point and vectors are projected along a normal vector onto the plane of each facet of the primitive. A parallelogram is formed with it's origin defined by the projected reference point. A vector (*vector1*) defines one side of the parallelogram. This vector's direction is parallel to the first projected reference vector and has a length that is equal to the absolute value of the x-component of the pattern size. The adjacent side of the parallelogram is defined similarly by the second projected reference vector (*vector2*) and the absolute value of the y-component of the pattern size. The pattern, a list of *m* x *n* equally-sized color cells, is applied across the parallelogram such that the first cell has the parallelogram origin in one corner. The first *m* colors in the pattern correspond to the row of cells nearest to and along the first projected reference vector (*vector1*). The pattern is mapped onto the facet by replicating the pattern parallelogram in directions parallel to the parallelogram's sides until the entire facet is covered.

If you specify the interior style as **PEXInteriorStylePattern**, then the implementation uses the pattern size width and height plus the pattern reference point and the pattern reference vector to position, scale, and rotate the pattern on the surface. If you specify either of the x-component and/or the y-component as zero, then the implementation ignores this output command and the pattern size entry of the pipeline state is not updated.

This function is a member of the styled and shaded surface attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, PEXOCCSetPatternAttributes

PEXOCCSetPickID - Set Pick Identifier (5.0)

Synopsis

void **PEXOCCSetPickID**(`PEXOCC` *context*, unsigned long *pick_id*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
pick_id	The pick identifier.

Returns

None.

Description

Invoke **PEXOCCSetPickID** to create an attribute output command which sets the current pick identifier attribute in the renderer's pipeline state according to the value specified by *pick_id*.

The pick identifier is returned as part of the path when you pick a primitive. When you are structure editing, you can use the pick identifier for navigation purposes.

Errors

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

See Also

PEXOCCSetPlaneMask - Set Plane Mask (5.2)

Synopsis

void **PEXOCCSetPlaneMask**(`PEXOCC` *context*, unsigned long *mask*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
mask	The plane mask.

Returns

None.

Description

Invoke **PEXOCCSetPlaneMask** to set the plane mask entry of the renderer's pipeline state according to the value you specify in *mask*.

The bits set to 1 in the plane mask determine which planes of the destination image buffer are affected by rendering requests. Each bit of the plane bitmask represents one plane of the image buffer. The least-significant bit of the plane bitmask (0x00000001) corresponds to the plane of a monochrome image buffer; additional planes are represented by the more significant bits. Specify the macro constant **AllPlanes** (defined in *Xlib.h*)to refer to all the planes.

The plane mask functions like the core X GC plane mask.

Except for all planes enabled, plane masking interacts with alpha blending, primitive antialiasing, transparency, and the drawing functions, except *GXcopy*, in an implementation-dependent way.

This function is a member of the global attribute set.

Errors

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

See Also

PEXOCCSetPolylineInterpMethod - Set Polyline Interpolation Method (5.0)

Synopsis

void **PEXOCCSetPolylineInterpMethod**(`PEXOCC` *context*, int *method*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
method	The polyline interpolation method (**PEXPolylineInterpNone** or **PEXPolylineInterpColor**).

Returns

None.

Description

Invoke **PEXOCCSetPolylineInterpMethod** to create an attribute output command which sets the polyline interpolation method attribute in the renderer's pipeline state according to the value specified by *method*.

By setting the polyline interpolation method attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's polyline interpolation method entry to specify how color values are computed along the segments when rendering polyline primitives. The interpolation method applies to all components of color, including alpha. When you set the polyline interpolation method attribute's ASF to **PEXBundled**, the implementation uses the polyline interpolation method from the line bundle table.

PEXlib defines the following polyline interpolation methods (*method*):

> **PEXPolylineInterpNone** - does not perform interpolation between polyline vertices. If you supply color values that differ for the endpoints of a polyline segment, then it is implementation-dependent whether the color of the i^{th} vertex is used to draw the line between the i^{th} and $(i+1)^{th}$ vertices (if n is the number of vertices, the color at the n^{th} vertex is ignored), or whether they are used to compute an average color which are applied to the entire segment.**(5.0)**
>
> **PEXPolylineInterpColor** - uses the polyline's vertex colors, if present. Then computes the color values along each polyline segment by linearly interpolating between the color values at the vertices.**(5.0)**

If the implementation does not support the specified interpolation *method*, then the method defaults to **PEXPolylineInterpNone**. Inquire the supported values for the *polyline interpolation method* by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the shaded line attribute set.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.

BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXOCCSetIndividualASF**

PEXOCCSetPrimitiveAntialiasing - Set Primitive Antialiasing (5.2)

Synopsis

void **PEXOCCSetPrimitiveAntialiasing**(`PEXOCC` *context*, unsigned int *prims*, int *blend_func*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
prims	The primitive types selected for antialiasing (**PEXPrimitiveAANone**, **PEXPrimitiveAAPoint**, **PEXPrimitiveAAVector**, **PEXPrimitiveAAPolygon**, **PEXPrimitiveAARasterText**, **PEXPrimitiveAAVectorText**, **PEXPrimitiveAAPolygonalText**).
blend_func	The blend function to use.

Returns

None.

Description

Invoke **PEXOCCSetPrimitiveAntialiasing** to create an attribute output command which sets the primitive antialiasing entry of the renderer's pipeline state according to the value specified by *prims*.

Primitive antialiasing specifies modifications made to the image on a type of primitive basis to correct for aliasing (jagged) effects. Typically the algorithms smooth edges through a blending operation.

PEXlib defines the following primitive types (*prims*):

PEXPrimitiveAANone - does not antialias primitives.**(5.2)**

PEXPrimitiveAAPoint - the set of primitives and attributes which combine to form a point primitive is implementation-dependent but at a minimum includes marker type **PEXMarkerDot** applied to **PEXOCCMarkers**.**(5.2)**

PEXPrimitiveAAVector - The set of primitives and attributes which combine to form a *vector* primitive is implementation-dependent but at a minimum includes **PEXOCCPolyline** and **PEXOCCPolylines**, all of nominal width.**(5.2)**

PEXPrimitiveAAPolygon - The set of primitives and attributes which combine to form a *polygon* primitive is implementation-dependent but at a minimum includes interior style *Solid* applied to:**(5.2)**

PEXOCCArcAreas
PEXOCCCones
PEXOCCFillArea
PEXOCCFillAreaSet
PEXOCCIndexedFillAreaSets
PEXOCCIndexedTriangles
PEXOCCNURBSurface
PEXOCCPolyTriangle
PEXOCCQuadrilateralMesh

PEXOCCSpheres
PEXOCCTriangleFan
PEXOCCTriangles
PEXOCCTriangleStrip

PEXPrimitiveAARasterText - **PEXText3D**, **PEXText2D**, **PEXAnnotationText3D**, and **PEXAnnotationText2D** with a raster font applied. **(5.2)**

PEXPrimitiveAAVectorText - **PEXText3D**, **PEXText2D**, **PEXAnnotationText3D**, and **PEXAnnotationText2D** with a stroked font applied.**(5.2)**

PEXPrimitiveAAPolygonalText - **PEXText3D**, **PEXText2D**, **PEXAnnotationText3D**, and **PEXAnnotationText2D** with a polygonal font applied.**(5.2)**

To determine which primitive types you can antialias, inquire *PrimAntiAliasSupported* with request **PEXGetImpDepConstants**. The implementation antialiases any primitives of the types selected by *prims.* It leaves aliased any primitives of the types selected by *prims* which it cannot antialias. If you specify an undefined value for *prims*, then the implementation issues a *BadPEXOutputCommand* error.

If the implementation is accomplishing primitive antialiasing through a blending function, then you have control over how the blending is accomplished. PEXlib defines the following values for the blending functions:

PEXPrimitiveAABlendOpImpDep - specifies an implementation-dependent blending function. **(5.2)**

PEXPrimitiveAABlendOpSimpleAlpha - specifies that an alpha value is calculated based on the primitive's coverage of the pixel and applied such that:**(5.2)**

$\text{Color} = \alpha * \text{Color}_{\text{primitive}} + (1 - \alpha) * \text{Color}_{\text{previous}}$

If the implementation does not support the value you specify in *blend_func*, then the blend function defaults to **PEXPrimitiveAABlendOpImpDep**. Inquire the supported values for the blend function by invoking the **PEXGetEnumTypeInfo** function.

Primitive antialiasing interacts with HLHSR, alpha blending, transparency, drawing functions (other than *GXcopy*), and plane masking (other than all planes enabled) in an implementation-dependent way.

This function is a member of the global attribute set.

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, **PEXGetImpDepConstants, PEXOCCSetIndividualASF**

PEXOCCSetReflectanceBundleIndex - Set Reflectance Bundle Index (5.2)

Synopsis

void **PEXOCCSetReflectanceBundleIndex**(`PEXOCC` *context*, `unsigned` int *index*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The reflectance bundle table index.

Returns

None.

Description

Invoke **PEXOCCSetReflectanceBundleIndex** to create an attribute output command which sets an index to an interior bundle table in the renderer’s pipeline state according to the value specified by *index*.

You use the reflection bundle index to select an entry of the renderer’s interior bundle table whenever a bundlable reflection attribute’s ASF is set to **PEXBundled** (see **PEXOCCSetIndividualASF**) and the renderer’s PHIGS interior bundle mode is **True**. If you set the PHIGS interior bundle mode to **False**, then the implementation uses the interior bundle index to select the entry of the interior bundle from which the bundled reflectance attributes are taken. PEXlib supports the bundling of the following reflectance attributes: surface reflectance model and surface reflectance attributes.

If you specify an undefined *index*, then the index defaults to index one. If index one is not defined, then the implementation sets the bundled surface reflectance model to **PEXReflectionNone**; the ambient, diffuse, and specular coefficients to 1.0; the specular concentration to 0.0; the specular color to indexed color one, and the transmission coefficient to 0.0. If you specify a reflectance bundle *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the basic surface attribute set.

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, **PEXOCCSetBFReflectanceBundleIndex**, **PEXOCCSetIndividualASF**

PEXOCCSetReflectionAttributes - Set Surface Reflection Attributes (5.0)

Synopsis

void **PEXOCCSetReflectionAttributes**(`PEXOCC` *context*, `PEXReflectionAttributes5_2` **attributes*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
attributes	The surface reflection attributes [§2.24.11, page 59], [§2.24.10, page 58].

Returns

None.

Description

Invoke **PEXOCCSetReflectionAttributes** to create an attribute output command which sets the surface reflection attributes in the renderer's pipeline state according to the value specified by *attributes*.

By setting the surface reflection attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's surface reflection attributes entry to define the reflection calculation used to light surface primitives. When you set the surface reflection attribute's ASF to **PEXBundled**, the implementation uses the surface reflection attributes from the interior bundle table.

If you do not set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**), then the implementation applies the specified surface reflection attributes to back-facing as well as front-facing surface interiors**.** Otherwise the implementation uses the back-facing surface reflection attributes (see **PEXOCCSetBFReflectionAttributes**).

The *attributes* argument defines surface reflection attributes that include ambient, diffuse and specular coefficients, the specular concentration and color, and the transmission coefficient. The ambient, diffuse, and specular coefficients define the fraction of ambient, diffuse, and specular light respectively reflected by a surface. Specify a value of 0.0 to reflect no light. Specify a value of 1.0 to reflect 100% of the light.

The larger the value of the *specular concentration* value of the reflection attributes, the shinier the surface.If the implementation does not support the specified color type, then the implementation uses a specular color type of **PEXColorTypeIndexed** with a value of 1. If you specify an indexed color with an index of 65535, then the implementation issues a *BadPEXOutputCommand* error. If you specify out of range direct color components, then the result is implementation-dependent.

Inquire whether the transmission coefficient is used in the reflection calculation by inquiring the transparency supported via the **PEXGetImpDepConstants** function. The transmission coefficient defines the default alpha component of surface primitives without an explicit alpha-component. Specify a transmission coefficient of 0.0 for an opaque surface. Specify a value of 1.0 for a transparent surface. The alpha-component value is determined by subtracting the transmission coefficient from 1.

This function is a member of the general surface attribute set.

Errors

BadPEXOutputCommand	The specified color index is 65535.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXGetImpDepConstants, **PEXOCCSetBFReflectionAttributes**, **PEXOCCSetIndividualASF**

PEXOCCSetReflectionModel - Set Surface Reflection Model (5.0)

Synopsis

void **PEXOCCSetReflectionModel**(`PEXOCC` *context*, int *model*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
model	The surface reflection model (**PEXRMNone**, **PEXRMAmbient**, **PEXRMAmbientDiffuse** or **PEXRMAmbientDiffuseSpecular**).

Returns

None.

Description

Invoke **PEXOCCSetReflectionModel** to create an attribute output command which sets the surface reflection model attribute in the renderer's pipeline state according to the value specified by *model*.

By setting the surface reflection models attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF),** you use the pipeline state's surface reflection model entry to specify which reflectance components to consider when performing reflectance calculations. When you set the surface reflection model attribute's ASF to **PEXBundled**, the implementation uses the surface reflection model from the interior bundle table.

If you do not set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**), then the implementation applies the specified surface reflection model to back-facing as well as front-facing surface interiors**.** Otherwise the implementation uses the back-facing surface reflection model (see **PEXOCCSetBFReflectionModel**).

PEX defines the following surface reflection models (*model*):

> **PEXRMNone** - performs no reflectance calculation. The surface color is not affected by light sources.**(5.0)**
>
> **PEXRMAmbient** - uses only the ambient terms of the reflectance calculation.**(5.0)**
>
> **PEXRMAmbientDiffuse** - uses only the ambient and diffuse terms of the reflectance calculation.**(5.0)**
>
> **PEXRMAmbientDiffuseSpecular** - uses the ambient, diffuse, and specular terms of the reflectance calculation.**(5.0)**

Note: The reflection models defined above simply have new names for PEXlib 5.2 and are completely equivalent to their respective names in PEXlib 5.1: **PEXReflectionNone**, **PEXReflectionAmbient**, **PEXReflectionDiffuse**, and **PEXReflectionSpecular**

If the implementation does not support the specified reflection *model*, then the model defaults to **PEXRMNone**. Inquire the supported values for the *surface reflection model* by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the general surface attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, **PEXOCCSetBFReflectionModel**, **PEXOCCSetIndividualASF**

PEXOCCSetRenderingColorModel - Set Rendering Color Model (5.0)

Synopsis

void **PEXOCCSetRenderingColorModel**(`PEXOCC` *context*, int *model*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
model	The rendering color model (**PEXRenderingColorModelImpDep**, **PEXRenderingColorModelRGB**, **PEXRenderingColorModelCIE**, **PEXRenderingColorModelHSV**, **PEXRenderingColorModelHLS**, or **PEXRenderingColorModelYIQ**).

Returns

None.

Description

Invoke **PEXOCCSetRenderingColorModel** to create an attribute output command which sets the rendering color model attribute in the renderer's pipeline state according to the value specified by *model*.

The rendering color model defines the color model actually used for the rendering calculations. You can define color values in color models other than rendering color model; PEXlib converts them to the rendering color model before using them.

PEXlib defines the following color models (*model*):

PEXRenderingColorModelImpDep - specifies an implementation-dependent color space.**(5.0)**

PEXRenderingColorModelRGB - specifies a red, green, blue color model.**(5.0)**

PEXRenderingColorModelCIE - specifies a CIELUV diagram u, v coordinates plus luminance color model.**(5.0)**

PEXRenderingColorModelHSV - specifies a hue, saturation, value color model.**(5.0)**

PEXRenderingColorModelHLS - specifies a hue, lightness, saturation color model.**(5.0)**

PEXRenderingColorModelYIQ - specifies a (NTSC) luminance (Y), inphase (wideband orange-cyan), and quadrature (narrowband magenta-green) color model. **(5.0)**

If the implementation does not support the specified color *model*, then model defaults to **PEXRenderingColorModelImpDep** (model 0). Inquire the supported values for the *rendering color model* by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the global attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo

PEXOCCSetSurfaceActiveTextures - Set Surface Active Textures (5.2)

Synopsis

void **PEXOCCSetSurfaceActiveTextures**(`PEXOCC` *context*, unsigned int *count*, `PEXTableIndex` **indices*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
count	The number of indices in the list.
indices	A list of table indices into the texture mapping lookup table [§2.3.21, page 29].

Returns

None.

Description

Invoke **PEXOCCSetSurfaceActiveTextures** to create an attribute output command which sets the surface active textures attribute in the renderer's pipeline state according to the values specified by *indices*.

You use the pipeline state's active texture indices as an entry into the renderer's texture mapping lookup table. The indices are an ordered list that you use to select the textures that are sequentially applied to textured surfaces. The active texture specifies the texture map and an associated set of values. It applies only to surface primitives when the primitive's interior style is **PEXInteriorStyleTexture** and the active textures field is non-empty.

If you do not set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**), then the implementation applies active textures to the facets of back-facing as well as front-facing surface interiors when the interior style is **PEXInteriorStyleTexture** and the active textures field is non-empty. Otherwise the implementation uses the back-facing surface active textures when the interior style is **PEXInteriorStyleTexture** and the active textures field is non-empty (see **PEXOCCSetBFSurfaceActiveTextures**).

If the number of indices you specify exceeds the maximum number of texture maps that your implementation can apply to a single surface, then the implementation ignores textures defined beyond the limit. Inquire the maximum number of texture maps, which you can apply to a single surface, by invoking the **PEXGetImpDepConstants** function.

If you specify a texture mapping lookup table *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the styled and shaded surface attribute set.

Errors

BadPEXOutputCommand	An index is zero.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXGetImpDepConstants, PEXOCCSetBFSurfaceActiveTextures

PEXOCCSetSurfaceApprox - Set Surface Approximation Method (5.0)

Synopsis

void **PEXOCCSetSurfaceApprox**(`PEXOCC` *context*, int *method*, double *utolerance*, double *vtolerance*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
method	The surface approximation method (**PEXApproxImpDep**, **PEXApproxConstantBetweenKnots**, **PEXApproxWCChordalSize**, **PEXApproxNPCChordalSize**, **PEXApproxDCChordalSize**, **PEXApproxWCPlanarDev**, **PEXApproxNPCPlanarDev**, **PEXApproxDCPlanarDev**, **PEXApproxWCRelative**, **PEXApproxNPCRelative** or **PEXApproxDCRelative**).
utolerance	The surface approximation tolerance in the *u* direction.
vtolerance	The surface approximation tolerance in the *v* direction.

Returns

None.

Description

Invoke **PEXOCCSetSurfaceApprox** to create an attribute output command which sets the surface approximation method attribute in the renderer's pipeline state according to the value specified by *method*.

By setting the surface approximation attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's surface approximation entry to specify the smoothing method to be used when rendering curved bivariate primitives. When you set the surface approximation attribute's ASF to **PEXBundled**, the implementation uses the surface approximation from the interior bundle table.

PEXlib defines the following *surface approximation methods* and the use of *utolerance* and *vtolerance*:

PEXApproxImpDep - smooths the surface in an implementation-dependent way and ignores the *utolerance* and *vtolerance*.**(5.0)**

PEXApproxConstantBetweenKnots - tessellates the surface with equal parametric increments between successive pairs of knots. **(5.0)**

utolerance and *vtolerance* controls the tessellation in each of the two parameter dimensions. If you specify a fraction for *utolerance or vtolerance*, then the implementation truncates it and only uses the integer portion. If you specify a *utolerance* or *vtolerance* that is less than or equal to zero, then the surface is evaluated only at the parameter limits and at the knots that are within the specified parameter range in that tolerance's direction. If *utolerance* or *vtolerance* is greater than zero, in the tolerance's direction, then the surface is evaluated at the parameter limits, at the knots that are within the specified parameter range, and at the number of positions specified by tolerance between each pair of knots.

PEXApproxWCChordalSize - tessellates the surface whereby the WC length of each segment (chord) in the *u* direction is less than the *utolerance* (specified in world coordinates) and each segment in the *v* direction is less than the *vtolerance* (specified in world coordinates).**(5.0)**

PEXApproxNPCChordalSize - tessellates the surface whereby the NPC length of each segment (chord) in the *u* direction is less than the *utolerance* (specified in normalized projection coordinates) and each segment in the *v* direction is less than the *vtolerance* (specified in normalized projection coordinates).**(5.0)**

PEXApproxDCChordalSize - tessellates the surface whereby the DC length of each segment (chord) in the *u* direction is less than the *utolerance* (specified in device coordinates) and each segment in the *v* direction is less than the *vtolerance* (specified in device coordinates).**(5.0)**

PEXApproxWCChordalDev - tessellates the surface whereby the deviation in WC's between the surface and its tessellated representation in the *u* direction does not exceed the *utolerance* (specified in world coordinates) and the tessellated representation in the *v* direction does not exceed the *vtolerance* (specified in world coordinates).**(5.0)**

PEXApproxNPCChordalDev - tessellates the surface whereby the deviation in NPC's between the surface and its tessellated representation in the *u* direction does not exceed the *utolerance* (specified in normalized projection coordinates) and the tessellated representation in the *v* direction does not exceed the *vtolerance* (specified in normalized projection coordinates). **(5.0)**

PEXApproxDCChordalDev - tessellates the surface whereby the deviation in DC's between the surface and its tessellated representation in the *u* direction does not exceed the *utolerance* (specified in device coordinates) and the tessellated representation in the *v* direction does not exceed the *vtolerance* (specified in device coordinates).**(5.0)**

PEXApproxWCRelative - indicates, with a *utolerance* value in the range [0.0, 1.0], a relative quality of rendering to be maintained independent of scaling in world coordinates (WCs). Specify a *utolerance* of 1.0 for the highest possible quality. The implementation ignores the *vtolerance*.**(5.0)**

PEXApproxNPCRelative - indicates, with a *utolerance* value in the range [0.0, 1.0], a relative quality of rendering to be maintained independent of scaling in normalized projection coordinates (NPC's). Specify a *utolerance* of 1.0 for the highest possible quality. The implementation ignores the *vtolerance*.**(5.0)**

PEXApproxDCRelative - indicates, with a *utolerance* value in the range [0.0, 1.0], a relative quality of rendering to be maintained independent of scaling in device coordinates (DC's). Specify a *utolerance* of 1.0 for the highest possible quality. The implementation ignores the *vtolerance*.**(5.0)**

If the implementation does not support the specified surface approximation *method*, then the method defaults to **PEXApproxImpDep**. Inquire the supported values for the *surface approximation method* by invoking the **PEXGetEnumTypeInfo** function.

Specify the *utolerance* and *vtolerance* to indicate the desired accuracy of the approximation. The surface approximation tolerance has different results depending on which method you specify. See *Chapter 4: Pipeline Contexts* for more information about tolerance values.

This function is a member of the curved surface attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, **PEXOCCSetIndividualASF**

PEXOCCSetSurfaceColor - Set Surface Color (5.0)

Synopsis

void **PEXOCCSetSurfaceColor**(`PEXOCC` *context*, `PEXColor` **color*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
color_type (OCC)	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
color	A pointer to the surface color [§2.5.12, page 32].

Returns

None.

Description

Invoke **PEXOCCSetSurfaceColor** to create an attribute output command which sets the surface color attribute in the renderer’s pipeline state according to the value specified by *color*.

By setting surface color attribute’s ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state’s surface color entry to define the surface color type. When you set the pipeline state’s surface color attribute’s ASF to **PEXBundled**, the implementation uses the surface color from the interior bundle table. If you do not set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**), then the implementation applies surface color to back-facing as well as front-facing surface interiors. Otherwise the implementation uses the back-facing surface color (see **PEXOCCSetBFSurfaceColor**).

Depending on the *color type* you specify, this function sets the attribute to either an indexed color or a direct color value. If you specify a *color type* of **PEXColorTypeIndexed**, but you specified an undefined index, then the index defaults to color index one. If color index one is not defined, then the surface color defaults to white. If you specified a *color index* greater than 65534, then the implementation issues a *BadPEXOutputCommand* error. If the *color type* is one of the supported direct color types and you specify an out of range component, then the result is implementation-dependent. If the color type is not supported, then indexed color one is used. Inquire the supported values for color type by invoking the **PEXGetEnumTypeInfo** function.

The surface color is a member of the styled and shaded surface attribute set.

Errors

BadPEXOutputCommand	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

PEXCreateLookupTable, **PEXGetEnumTypeInfo, PEXOCCSetBFSurfaceColor**, **PEXOCCSetIndividualASF**

PEXOCCSetSurfaceColorIndex - Set Surface Color Index (5.0)

Synopsis

void **PEXOCCSetSurfaceColorIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The color table index surfaces.

Returns

None.

Description

Invoke **PEXOCCSetSurfaceColorIndex** to set the surface color entry to an indexed color in the renderer's pipeline state according to the value specified by *index*. The index selects an entry of the renderer's color table. You can alternately use the **PEXOCCSetSurfaceColor** function which allows you to specify a color of any type.

By setting the surface color index attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's surface color entry to determine the color of surface attributes. When you set the surface color attribute's ASF to **PEXBundled**, the implementation uses the surface color from the interior bundle table.

If you do not set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**), then the implementation applies the surface color index to back-facing as well as front-facing surface interiors. Otherwise the implementation uses the back-facing surface color index (see **PEXOCCSetBFSurfaceColorIndex**).

If you specify an undefined color *index*, then the index defaults to color index one. If color index one is not defined, then the color defaults to white. If you specify a color *index* which is greater than 65534, then the implementation issues a *BadPEXOutputCommand* error.

The surface color index function is a member of the styled and shaded surface attribute set.

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

PEXCreateLookupTable, **PEXOCCSetBFSurfaceColorIndex**, **PEXOCCSetIndividualASF, PEXOCCSetSurfaceColor**

PEXOCCSetSurfaceEdgeColor - Set Surface Edge Color (5.0)

Synopsis

void **PEXOCCSetSurfaceEdgeColor**(`PEXOCC` *context*, `PEXColor` **color*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
color_type (OCC)	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
color	A pointer to the surface edge color [§2.5.12, page 32].

Returns

None.

Description

Invoke **PEXOCCSetSurfaceEdgeColor** to create an attribute output command which sets the surface edge color attribute in the renderer's pipeline state according to the value specified by *color*.

By setting the surface edge color attribute's ASF to **PEXBundled** (see **PEXOCCSetIndividualASF**), you can specify the color for rendering edges.

Depending on the *color type* you specify, this function sets the attribute to either an indexed color or a direct color value. If you specify a *color type* of PEX**ColorTypeIndexed**, but you specified an undefined index, then the index defaults to color index one. If color index one is not defined, then the surface edge color defaults to white. If you specified a *color index* greater than 65534, then the implementation issues a *BadPEXOutputCommand* error. If the *color type* is one of the supported direct color types and you specify an out of range component, then the result is implementation-dependent. If the *color type* is not supported, then indexed color one is used. Inquire the supported values for color type by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the edge attribute set.

Errors

BadPEXOutputCommand	The 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

PEXCreateLookupTable, **PEXGetEnumTypeInfo, PEXOCCSetIndividualASF**

PEXOCCSetSurfaceEdgeColorIndex - Set Surface Edge Color Index (5.0)

Synopsis

void **PEXOCCSetSurfaceEdgeColorIndex**(PEXOCC *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The color table index for surface edges.

Returns

None.

Description

Invoke **PEXOCCSetSurfaceEdgeColorIndex** to set the surface edge color entry to an indexed color in the renderer’s pipeline state according to the value specified by *index*. The index selects an entry of the renderer’s color table. You can alternately use **PEXOCCSetSurfaceEdgeColor** which allows you to specify a color of any type.

By setting the edge color index attribute’s ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state’s surface edge color entry to determine the color of surface edge attributes. When you set the surface edge attribute’s ASF to **PEXBundled**, the implementation uses the surface edge color from the edge bundle table.

If you specify an undefined color *index*, then the index defaults to color index one. If color index one is not defined, then color defaults to white. If you specify a color *index* which is greater than 65534, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the edge attribute set.

Errors

BadPEXOutputCommand	The 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

PEXCreateLookupTable, **PEXOCCSetIndividualASF, PEXOCCSetSurfaceEdgeColor**

PEXOCCSetSurfaceEdgeFlag - Set Surface Edge Flag (5.0)

Synopsis

void **PEXOCCSetSurfaceEdgeFlag**(PEXOCC *context*, int *flag*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	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

Invoke **PEXOCCSetSurfaceEdgeFlag** to create an attribute output command which sets the surface edge flag attribute in the renderer's pipeline state according to the value specified by *flag*.

By setting the surface edge attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's surface edge flag entry to indicate whether or not surfaces edges are drawn. When you set the surface edge attribute's ASF to **PEXBundled**, the implementation uses the surface edge flag from the edge bundle table.

PEXlib defines the following values for the surface edge flag (*flag*):

PEXOn - enables surface edges.**(5.0)**

PEXOff - disables surface edges. **(5.0)**

If you specify a *flag* value that is not defined, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the edge attribute set.

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

PEXOCCSetIndividualASF

PEXOCCSetSurfaceEdgeType - Set Surface Edge Type (5.0)

Synopsis

void **PEXOCCSetSurfaceEdgeType**(`PEXOCC` *context*, int *edge_type*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
edge_type	The surface edge type (**PEXSurfaceEdgeSolid**, **PEXSurfaceEdgeDashed**, **PEXSurfaceEdgeDotted** or **PEXSurfaceEdgeDashDot**).

Returns

None.

Description

Invoke **PEXOCCSetSurfaceEdgeType** to create an attribute output command which sets the surface edge type attribute in the renderer's pipeline state according to the value specified by *edge_type*.

By setting the surface edge type attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's surface edge type entry to describe the style used when rendering the edges of surface primitives. When you set the surface edge type attribute's ASF to **PEXBundled**, the implementation uses the surface edge type from the edge bundle table.

PEXlib defines the following edge types (*edge_type*):

> **PEXSurfaceEdgeSolid** - draws the surface edge with a solid, unbroken line.**(5.0)**
>
> **PEXSurfaceEdgeDashed** - draws the surface edge with a line that is broken into even segments.**(5.0)**
>
> **PEXSurfaceEdgeDotted** - draws the surface edge with a line that is broken into dots.**(5.0)**
>
> **PEXSurfaceEdgeDashDot** - draws the surface edge with a line that contains alternating dotted and dashed line segments.**(5.0)**

Whether edge types, except **PEXSurfaceEdgeSolid**, are restarted or continued at the start of the edge, at the start of a clipped segment of an edge, or at each vertex of a surface edge is implementation-dependent.

If the implementation does not support the specified edge type (*edge_type*), then the edge type defaults to **PEXSurfaceEdgeSolid**. Inquire the supported values for the *surface edge type* by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the edge attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, **PEXOCCSetIndividualASF**

PEXOCCSetSurfaceEdgeWidth - Set Surface Edge Width (5.0)

Synopsis

void **PEXOCCSetSurfaceEdgeWidth**(`PEXOCC` *context*, double *width*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
width	The surface edge width scale factor.

Returns

None.

Description

Invoke **PEXOCCSetSurfaceEdgeWidth** to create an attribute output command which sets the surface edge width attribute in the renderer’s pipeline state according to the value specified by *width*.

By setting the surface edge width attribute’s ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state’s surface edge width entry to define the surface edge width factor. When you set the surface edge width attribute’s ASF to **PEXBundled**, the implementation uses the surface edge width from the edge bundle table. When you render a surface primitive, the nominal edge width is multiplied by the surface edge width factor. Use this scale factor to increase or decrease the width (in pixels) from the nominal edge width for the display device. PEXlib maps the result to the nearest supported edge width. A negative surface edge width factor results in the thinnest available edge.

Inquire the nominal edge width, the minimum edge width, the maximum edge width, and the number of supported edge widths supported by invoking the **PEXGetImpDepConstants** request.

This function is a member of the edge attribute set.

Errors

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

See Also

PEXGetImpDepConstants, **PEXOCCSetIndividualASF**

PEXOCCSetSurfaceInterpMethod - Set Surface Interpolation Method (5.0)

Synopsis

void **PEXOCCSetSurfaceInterpMethod**(`PEXOCC` *context*, int *method*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
method	The surface interpolation method (**PEXSurfaceInterpNone**, **PEXSurfaceInterpColor**, **PEXSurfaceInterpDotProduct** or **PEXSurfaceInterpNormal**).

Returns

None.

Description

Invoke **PEXOCCSetSurfaceInterpMethod** to create an attribute output command which sets the surface interpolation method attribute in the renderer's pipeline state according to the value specified by *method*.

By setting the surface interpolation method attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's surface interpolation method entry to specify how color values, including any alpha components, are computed in the interior of surface primitives. When you set the surface interpolation method attribute's ASF to **PEXBundled**, the implementation uses the surface interpolation method from the interior bundle table.

If you do not set the facet distinguish flag to **True** (see **PEXOCCSetFacetDistinguishFlag**), then the implementation applies the surface interpolation method to back-facing as well as front-facing surface interiors. Otherwise the implementation uses the back-facing surface interpolation method (see **PEXOCCSetBFSurfaceInterpMethod**).

PEXlib defines the following surface interpolation methods (*method*):

PEXSurfaceInterpNone - uses the color resulting from a single evaluation of the reflectance calculation to render the entire surface. Interpolation is *not* performed across surface interiors or edges.**(5.0)**

PEXSurfaceInterpColor - performs the reflection calculation at the vertices of the surface. Then, linearly interpolates the resulting color values across the interior of the surface and the edges.**(5.0)**

PEXSurfaceInterpDotProduct - computes at each vertex any dot products needed by the reflectance calculation. Then, linearly interpolates these dot products and performs the reflectance calculation using these values to compute the color value at each pixel in the interior of a surface or along a surface edge.**(5.0)**

PEXSurfaceInterpNormal - interpolates vertex reflectance normals and performs reflectance calculations at each pixel in the interior of a surface or along a surface edge.**(5.0)**

If the implementation does not support the specified interpolation *method*, then the method defaults to **PEXSurfaceInterpNone**. Inquire the supported values for the *surface interpolation method* by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the styled and shaded surface attribute set.

Errors

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

See Also

PEXGetEnumTypeInfo, **PEXOCCSetBFSurfaceInterpMethod, PEXOCCSetIndividualASF**

PEXOCCSetTextAlignment - Set Text Alignment (5.0)

Synopsis

void **PEXOCCSetTextAlignment**(PEXOCC *context*, int *halignment*, int *valignment*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
halignment	The horizontal text alignment (**PEXHAlignNormal**, **PEXHAlignLeft**, **PEXHAlignCenter** or **PEXHAlignRight**).
valignment	The vertical text alignment (**PEXVAlignNormal**, **PEXVAlignTop**, **PEXVAlignCap**, **PEXVAlignHalf**, **PEXVAlignBase** or **PEXVAlignBottom**).

Returns

None.

Description

Invoke **PEXOCCSetTextAlignment** to create an attribute output command which sets the horizontal and vertical text alignment attributes in the renderer's pipeline state according to the values specified by *halignment* and *valignment* respectively.

The text alignment positions the text primitive's extent relative to the text position. The extent is an imaginary parallelogram which has sides that parallel the *x* and *y* axes of the text's coordinate system. The sides are bounded by the characters of the character string according to the font.

PEXlib defines the following horizontal text alignments (*halignment*):

PEXHAlignNormal - aligns the text with the left-side of the character string if you specify a text path of **PEXPathRight**. If you specify an a text path of **PEXPathLeft**, then the alignment is based on the right-side of the character string. If you specify a text path of **PEXPathUp** or **PEXPathDown**, then the alignment is based on the center of the character string.**(5.0)**

PEXHAlignLeft - aligns the text whereby the left-side of the extent passes through the text position.**(5.0)**

PEXHAlignCenter - aligns the text whereby the text position lies midway between the left and right sides of the text extent.**(5.0)**

PEXHAlignRight - aligns the text whereby the right-side of the extent passes through the text position.**(5.0)**

PEXlib defines the following vertical text alignments (*valignment*):

PEXVAlignNormal - aligns the text with the base if you specify a text path of **PEXPathRight**, **PEXPathLeft**, or **PEXPathUp**. If you specify a text path of **PEXPathDown**, then the alignment is based on the top of the text primitive's extent.

PEXVAlignTop - aligns the text whereby the top of the extent passes through the text position**.(5.0)**

PEXVAlignCap - aligns the text whereby the capline of the character string passes through the text position if you specify a text path of **PEXPathRight** or **PEXPathLeft.** If you specify a text path of **PEXPathUp or PEXPathDown,** then the alignment is based such that the capline of the top-most character in the string passes through the text position**.(5.0)**

The capline is a horizontal line which bounds the top of the characters of the character string according to the font. In general there is space between the capline and the topline for placing additional rendering, such as an accent, above the character string.

PEXVAlignHalf - aligns the text whereby the halfline of the character string passes through the text position if you specify a text path of **PEXPathRight** or **PEXPathLeft**. If you specify a text path of **PEXPathDown** or **PEXPathUp**, then the alignment is based such that the text position lies halfway between the halflines of the top and bottom characters.**(5.0)**

The halfline is a horizontal line which lies halfway between the capline and the baseline according to the font.

PEXVAlignBase - aligns the text whereby the baseline of the string passes through the text position if you specify a text path of **PEXPathRight** or **PEXPathLeft**. If you specify a text path of **PEXPathUp** or **PEXPathDown**, then the alignment is based such that the bottom line of the bottom-most portion of the character in the character string passes through the text position.**(5.0)**

The baseline is a horizontal line upon which the characters sit according to the font. In general there is space between the baseline and the bottomline for placing the lower portions of letters, such as the stems of the lowercase "j" and "q."

PEXVAlignBottom - aligns the text whereby the bottom of the extent passes through the text position.**(5.0)**

If you specify an undefined *text alignment*, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the text attribute set.

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

PEXOCCSetATextAlignment

PEXOCCSetTextBundleIndex - Set Text Bundle Index (5.0)

Synopsis

void **PEXOCCSetTextBundleIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The text bundle table index.

Returns

None.

Description

Invoke **PEXOCCSetTextBundleIndex** to create an attribute output command which sets the text bundle index attribute in the renderer's pipeline state according to the value specified by *index*.

You use the text bundle index to select an entry of the renderer's text bundle table whenever a bundlable text attribute's ASF is set to **PEXBundled** (see **PEXOCCSetIndividualASF**). PEXlib supports the bundling of the following text attributes: text font index, text precision, character expansion, character spacing, and text color.

If you specify an undefined text bundle *index*, then the index defaults to index one. If index one is not defined, then the implementation sets the bundled text font index to one, text precision to **PEXStringPrecision**; character expansion to 1.0, character spacing to 0.0 and text color to indexed color one. If you specify a text bundle *index* of zero, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the annotation and text attribute set.

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, **PEXOCCSetIndividualASF**

PEXOCCSetTextColor - Set Text Color (5.0)

Synopsis

void **PEXOCCetTextColor**(`PEXOCC` *context*, `PEXColor` **color*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
color_type (OCC)	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8**, or **PEXColorTypeRGBA16**).
color	A pointer to the text color [§2.5.12, page 32].

Returns

None.

Description

Invoke **PEXOCCSetTextColor** to create an attribute output command which sets the text color attribute in the renderer's pipeline state according to the value specified by *color.*

By setting the text color attribute's ASF to **PEXIndividual**, you use the pipeline state's text color to define the color applied to text. When you set the text color attribute's ASF to **PEXBundled**, the implementation uses the text color from the text bundled table.

Depending on the *color type* you specify, this function sets the attribute to either an indexed color or a direct color value. If you specify a *color type* of **PEXColorTypeIndexed**, but you specified an undefined index, then the index defaults to color index one. If color index one is not defined, then the text color defaults to white. If you specified a *color index* greater than 65534, then the implementation issues a *BadPEXOutputCommand* error. If the *color type* is one of the supported direct color types and you specify an out of range component, then the result is implementation-dependent. If the *color type* is not supported, then indexed color one is used. Inquire the supported values for color type by invoking the **PEXGetEnumTypeInfo** function.

This function is a member of the annotation and text attribute set.

Errors

BadPEXOutputCommand	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

PEXCreateLookupTable, **PEXGetEnumTypeInfo, PEXOCCSetIndividualASF**

PEXOCCSetTextColorIndex - Set Text Color Index (5.0)

Synopsis

void **PEXOCCSetTextColorIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The color table index for text.

Returns

None.

Description

Invoke **PEXOCCSetTextColorIndex** to set the text color attribute to an indexed color in the renderer's pipeline state according to the value specified by *index*. The index selects an entry of the renderer's color table. You can alternately use **PEXOCCSetTextColor** which allows you to specify a color of any type.

By setting the text color attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's text color entry to determine the color of text attributes. When you set the text color attribute's ASF to **PEXBundled**, the implementation uses the text color from the text bundle table.

If you specify an undefined color *index*, then the index defaults to color index one. If color index one is not defined, then the color defaults to white. If you specify a color *index* which is greater than 65534, then the implementation issues a *BadPEXOutputCommand* error.

The text color index function is a member of the annotation and text attribute set.

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

PEXCreateLookupTable, **PEXOCCSetIndividualASF, PEXOCCSetTextColor**

PEXOCCSetTextFontIndex - Set Text Font Index (5.0)

Synopsis

void **PEXOCCSetTextFontIndex**(PEXOCC *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The text font table index.

Returns

None.

Description

Invoke **PEXOCCSetTextFontIndex** to create an attribute output command which sets the text font index attribute in the renderer's pipeline state according to the value specified by *index*.

You use the text font index attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), to define the text font index into the text font table. Specify *index* to indicate the entry in the current text font lookup table that the implementation uses to render text and annotation text primitives. Each entry of the text font table defines a font group.

If you specify an undefined text font table *index*, then the index defaults to index one. If index one is not defined, then the result is implementation-dependent. If you specify a text font table *index* less than 1, then the implementation issues a *BadPEXOutputCommand* error.

If you attempt to use a font which is not available in the selected font group, then the text or annotation text primitive is rendered with the default font group. If a needed font is not available in the default font group, then the portion of the primitive specifying that font is rendered in an implementation-dependent way.

This function is a member of the annotation and text attribute set.

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, PEXOCCSetIndividualASF

PEXOCCSetTextPath - Set Text Path (5.0)

Synopsis

void **PEXOCCSetTextPath**(`PEXOCC` *context*, int *path*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
path	The text (drawing) path (**PEXPathRight**, **PEXPathLeft**, **PEXPathUp** or **PEXPathDown**).

Returns

None.

Description

Invoke **PEXOCCSetTextPath** to create an attribute output command which sets the text path attribute in the renderer's pipeline state according to the value specified by *path*.

Use the text path to define the direction in which you want text characters placed.

PEXlib defines the following text paths (*path*):

PEXPathRight - places each character after the previous character in a character string along a baseline in the direction of the character base vector.**(5.0)**

PEXPathLeft - places each character after the previous character in a character string along a baseline in the opposite direction of the character base vector.**(5.0)**

PEXPathUp - places each character after the previous character in a character string in the direction of the character up vector.**(5.0)**

PEXPathDown - places each character after the previous character in a character string in the opposite direction of the character up vector.**(5.0)**

If you specify an undefined *path*, then the implementation issues a *BadPEXOutputCommand* error.

The text path function is a member of the text attribute set.

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

PEXOCCSetATextPath

PEXOCCSetTextPrecision - Set Text Precision (5.0)

Synopsis

void **PEXOCCSetTextPrecision**(`PEXOCC` *context*, int *precision*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
precision	The text precision (**PEXStringPrecision**, **PEXCharPrecision** or **PEXStrokePrecision**).

Returns

None.

Description

Invoke **PEXOCCSetTextPrecision** to create an attribute output command which sets the text precision attribute in the renderer's pipeline state according to the value specified by *precision*.

By setting the text precision attribute's ASF to **PEXIndividual** (see **PEXOCCSetIndividualASF**), you use the pipeline state's text precision entry to define how precisely the implementation applies attributes to text (regular and annotated) and clips the text. When you set the text precision attribute's ASF to **PEXBundled**, the implementation uses the text precision from the text bundle table.

PEXlib defines the following text precisions:

> **PEXStringPrecision** - applies the text font index, text color, character height, and character expansion attributes to text primitives.**(5.0)**
>
> The character up vector, text path, text alignment, and character spacing attributes are not necessarily applied to text primitives.
>
> Performs clipping in an implementation-dependent way.
>
> **PEXCharPrecision** - applies all text attributes to text primitives.**(5.0)**
>
> Performs clipping at least on a character-by-character basis.
>
> **PEXStrokePrecision** - applies all text attributes to text primitives.**(5.0)**
>
> Performs clipping exactly at the clipping boundaries.

An implementation may substitute any text precision as a better text precision, where **PEXStringPrecision** is the least precise and **PEXStrokePrecision** is the best precision.

When text or annotation text is interpreted, the implementation renders the mono-encodings in the text string in the same text precision (least common denominator). If the font group selected by the current text font index consists of both X and PEX fonts, and if some of the mono-encodings in the string are rendered in X fonts, then the text precision of the entire string must be dropped to at least **PEXCharPrecision**.

If you specify an undefined *precision*, then the implementation issues a *BadPEXOutputCommand* error.

This function is a member of the annotation and text attribute set.

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

PEXOCCSetIndividualASF

PEXOCCSetTMPerspectiveCorrection - Set Texture Mapping Perspective Correction (5.2)

Synopsis

void **PEXOCCSetTMPerspectiveCorrection**(`PEXOCC` *context*, int *method*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
method	The type of texture mapping perspective correction to apply (**PEXTMPerspectiveCorrectionNone**, **PEXTMPerspectiveCorrectionVertex**, or **PEXTMPerspectiveCorrectionPixel**).

Returns

None.

Description

Invoke **PEXOCCSetTMPerspectiveCorrection** to create an attribute output command which sets the texture mapping perspective correction entry of the renderer's pipeline state according to the value specified by *method*.

The texture mapping perspective correction specifies if and how texture coordinates are adjusted for a perspective view mapping projection when rendering textured surface primitives.

PEXlib defines the following texture mapping perspective correction methods:

PEXTMPerspectiveCorrectionNone - linearly interpolates texture coordinates without any effort to apply correction or perspective view mapping.**(5.2)**

PEXTMPerspectiveCorrectionVertex - manipulates the texture coordinates associated with each vertex to account for perspective view mapping. Then, linearly interpolates the corrected values between vertices.**(5.2)**

PEXTMPerspectiveCorrectionPixel - manipulates the texture coordinate values at each step as texture coordinates are interpolated to account for perspective view mapping.**(5.2)**

If you specify an unsupported texture mapping perspective correction *method*, then the method used is implementation-dependent.

The texture mapping perspective correction function is a member of the styled and shaded surface attribute set.

Errors

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

See Also

PEXOCCSetTMResourceHints - Set Texture Mapping Resource Hints (5.2)

Synopsis

void **PEXOCCSetTMResourceHints**(`PEXOCC` *context*, int *optimization_hint*, unsigned int *count*, `PEXTableIndex` **priorities*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
optimization_hint	The resource optimization approach to consider for all subsequently activated textures (**PEXTMResourceHintsNone**, **PEXTMResourceHintsSpeed**, or **PEXTMResourceHintsSpace**).
count	The number of indices.
priorities	An array of texture binding table indices [§2.3.21, page 29].

Returns

None.

Description

Invoke **PEXOCCSetTMResourceHints** to create an attribute output command which sets the optimization hint in the renderer's pipeline state according to the value specified by *optimization_hint*.

Use the texture mapping resource hint to specify how the implementation should consider optimizing its resources when subsequent entries of the texture mapping lookup table are activated.

PEXlib defines the following texture mapping resource hints (*optimization_hint*):

> **PEXTMResourceHintsNone** - specifies that no consideration should be made with respect to optimization.**(5.2)**
>
> **PEXTMResourceHintsSpeed** - specifies that speed should be considered first with respect to optimization. **(5.2)**
>
> **PEXTMResourceHintsSpace** - specifies that space should be considered first with respect to optimization.**(5.2)**

The pipeline state's texture mapping resource hints also includes a prioritized list of indices into the renderer's texture mapping lookup table. This list indicates the expected usage of the indicated textures. If the implementation uses the list, it only applies to subsequently activated textures, both when they are activated and when they are deactivated. Textures already activated and subsequently activated textures that are not in the list are assumed to be at some implementation-dependent priority lower than the textures in the list. The first texture in the list is expected to be activated most often, and the last texture the least, of the specified textures. If the same index appears multiple times in the list, then its highest priority is used. If an index is undefined, then it is ignored. The prioritized textures replace the list in the pipeline state's texture mapping resource hints.

If you specify an unsupported texture mapping resource optimization hint (*priorities*), then the optimization hint used is **PEXTMResourceHintsNone**. If *priorities* includes index zero, then the implementation issues a *BadPEXOutputCommand* error.

The texture mapping resource hints function is a member of the styled and shaded surface attribute set.

Errors

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

See Also

PEXOCCSetTMSampleFrequency - Set Texture Mapping Sample Frequency (5.2)

Synopsis

void **PEXOCCSetTMSampleFrequency**(PEXOCC *context*, int *frequency*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
frequency	The texture mapping sample frequency (**PEXTMSampleFrequencyPixel** or **PEXTMSampleFrequencyInterpDep**).

Returns

None.

Description

Invoke **PEXOCCSetTMSampleFrequency** to create an attribute output command which sets the texture mapping sample frequency entry of the renderer's pipeline state according to the value specified by *frequency*.

Use the texture mapping sample frequency to specify the frequency to sample texels in a texture map when rendering textured surface primitives.

PEXlib defines the following texture mapping sample frequencies (*frequency*):

> **PEXTMSampleFrequencyPixel** -samples texture map texels once for each pixel. **(5.2)**
>
> **PEXTMSampleFrequencyInterpDep** - the texture map sample frequency is dependent on the pipeline state's surface interpolation method (see **PEXOCCSetSurfaceInterpMethod**) as follows:**(5.2)**
>
> If the interpolation method is **PEXSurfaceInterpNone**, then the implementation samples texture map texels once per facet.
>
> If the interpolation method is **PEXSurfaceInterpColor**, then the implementation samples texture map texels once per vertex.
>
> If the interpolation method is **PEXSurfaceInterpDot** or **PEXSurfaceInterpNormal**, then the implementation samples texture map texels once per pixel.

If you specify an unsupported texture mapping sample *frequency*, then the frequency defaults to **PEXSampleFrequencyPixel**.

The texture mapping sample frequency function is a member of the styled and shaded surface attribute set.

Errors

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

See Also

PEXOCCSetBFSurfaceInterpMethod, PEXOCCSetSurfaceInterpMethod

PEXOCCSetVertexOrder - Set Vertex Order (5.2)

Synopsis

void **PEXOCCSetVertexOrder**(PEXOCC *context*, unsigned int *order*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
order	The vertex order (**PEXVertexOrderRightHand** or **PEXVertexOrderLeftHand**).

Returns

None.

Description

Invoke **PEXOCCSetVertexOrder** to set the vertex order entry of the renderer's pipeline state according to the value specified by *order*.

PEXlib uses the vertex order in computing the facet normals of surfaces. For surface primitives which do not have facet normals supplied, the normal for each facet is calculated assuming the vertex order given through *order*.

If the vertex *order* is not set to **PEXVertexOrderRightHand** or **PEXVertexOrderLeftHand**, then the implementation issues a *BadPEXOutputCommand* error.

The vertex order function is a member of the general surface attribute set.

Errors

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

See Also

PEXOCCSetViewIndex - Set View Index (5.0)

Synopsis

void **PEXOCCSetViewIndex**(`PEXOCC` *context*, unsigned int *index*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
index	The view table index.

Returns

None.

Description

Invoke **PEXOCCSetViewInde**x to create an attribute output command which sets the view index attribute in the renderer's pipeline state according to the value specified by *index*.

Use the view index to select an entry of the renderer's view table. The view defines a volume in world coordinates being viewed, from where it is viewed, and how it is placed in normalized projection coordinate (NPC) space.

If you specify an undefined view index, then the index defaults to view table index zero. If view index zero is not defined, then the implementation sets all clip flags to **PEXClip**, the clip limits to <0,0,0>, <1,1,1>; the orientation and mapping matrices to identity, and the eyepoint to **PEXViewEyepointImplicit**.

If the *index* you specify exceeds 65534, then the implementation issues a *BadPEXOutputCommand* error.

The view index function is a member of the global attribute set.

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

PEXOCCSetZBufferTest - Set Z Buffer Test Criteria (5.2)

Synopsis

void **PEXOCCSetZBufferTest**(PEXOCC *context*, int *z_test*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
z_test	The Z buffer test criteria (**PEXZBufferTestLessThan**, **PEXZBufferTestLessThanOrEqual**, **PEXZBufferTestGreaterThan**, **PEXZBufferTestGreaterThanOrEqual**, **PEXZBufferTestEqual**, **PEXZBufferTestNotEqual**, or **PEXZBufferTestPass**).

Returns

None.

Description

Invoke **PEXOCCSetZBufferTest** to set the Zbuffer test entry of the renderer's pipeline state according to the value specified by *z_test*.

The z-buffer test specifies the criterion used to determine whether a pixel is visible or hidden when you enable HLHSR and accomplish it via z-buffering. *z_test* defines how the pixel's z-component is compared against the corresponding z-buffer value. If the comparison is **True**, then the pixel is visible. The z-buffer test is applied in a right-handed coordinate system where a larger z-value is closer to the eyepoint.

PEXlib defines the following z-buffer tests (z-*test*):

PEXZBufferTestLessThan - if pixel z value < z-buffer value, then the pixel is visible **(5.2)**

PEXZBufferTestLessThanOrEqual - if pixel z value <= z-buffer value, then the pixel is visible **(5.2)**

PEXZBufferTestGreaterThan - if pixel z value > z-buffer value, then the pixel is visible **(5.2)**

PEXZBufferTestGreaterThanOrEqual - if pixel z value >= z-buffer value, then the pixel is visible **(5.2)**

PEXZBufferTestEqual - if pixel z value == z-buffer value, then the pixel is visible **(5.2)**

PEXZBufferTestNotEqual - if pixel z value != z-buffer value, then the pixel is visible **(5.2)**

PEXZBufferTestPass - the pixel is always visible **(5.2)**

If you specify an unsupported z-buffer test criteria value (*z_test*), then the z-buffer test used is implementation-dependent.

This function is a member of the global attribute set.

Errors

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

See Also

Chapter 9

Structure Control and Miscellaneous Functions

There are some output commands that control the traversal of structure resources or do not directly cause the implementation to render anything.

The following Structure Control output commands control the renderer's traversal of structure networks:

"PEXOCCConditionFlags - Set Condition Flags (5.2)" on page 424

"PEXOCCConditionalExecuteStructure - Conditional Execute Structure (5.2)" on page 425

"PEXOCCConditionalIncludeStructure - Conditional Include Structure (5.2)" on page 426

"PEXOCCConditionalReturn - Conditional Return (5.2)" on page 427

"PEXOCCExecuteStructure - Execute Structure (5.0)" on page 428

"PEXOCCIncludeStructure - Include Structure (5.2)" on page 429

"PEXOCCTestExtent - Test Extent Output Command (5.2)" on page 434

"PEXOCCTestExtent2D - Test Extent 2D Output Command (5.2)" on page 436

The following miscellaneous output commands do not cause the renderer to render anything. They are provided primarily for application convenience:

"PEXOCCApplicationData - Structure Application Data (5.0)" on page 423

"PEXOCCGSE - Generalized Structure Element (5.0)" on page 430

"PEXOCCLabel - Structure Label (5.0)" on page 431

"PEXOCCNoop - Noop Output Command (5.1)" on page 432

Another output command controls the degree of parallelism that a renderer may use when processing the structure:

"PEXOCCSequentialProcessing - Sequential Processing of Output Commands (5.2)" on page 433

9.1. Conditional Traversal

Several of the output commands described in this chapter, along with the Extent Size lookup table, team up to provide a useful facility known as conditional traversal. One common use of conditional traversal is to reduce the amount of graphic data sent to the renderer if the graphic data is either out of view (clip test) or too small to be useful (size test).

9.1.1. Clip Test

Although the renderer will clip out-of-view data, it is often faster for the application to communicate the extent of an entire sub-network to the renderer and let the renderer perform the clip test once for the entire sub-network. Applications often keep graphic data that is contained in a small volume within a structure or sub-network.

To take advantage of this information, use the Test Extent output command to tell the renderer what the extent is of an object stored in a structure or sub-network. The renderer determines if the object is clipped completely from the view volume and sets a condition flag to indicate the result. Next, use the Conditional Execute Structure or Conditional

Include Structure to draw the structure or sub-network depending on the condition flag. If the condition flag indicates that the object was completely clipped, the renderer does not traverse the requested sub-network, otherwise it does traverse it.

Note that this procedure saved the renderer the task of clip testing each primitive in the sub-network in the case when the entire object was clipped.

9.1.2. Size Test

In some complex scenes, very detailed objects may only be a few pixels in size due to the current transformations. It is often a waste to draw the entire object because the details are lost in the small number of pixels the object occupies on the screen. It is therefore desirable to draw an alternate description of the object that represents the object with less detail and is faster to draw.

The principle is similar to the clip test, except that the renderer compares the supplied extent to the value given in the Extent Size lookup table. The Test Extent OC's use a lookup table instead of a direct value to allow the application to alter all the values in the table to easily control the thresholds at which these size decisions are made. The renderer transforms the extent to DC before comparing the extent to the value in the Extent Size table, which is also in DC. The renderer stores the results of the test in the condition flags. You then use the three conditional output commands to control drawing of the various representations of the object.

Example: Suppose you want to draw a scene with many houses. The user is able to change the view so that the houses are displayed in any size, from very large to very small. If the house is displayed larger than a certain size (> 20 pixels), you want to draw the outline of the house, and if the house is smaller, then a dot polymarker will do. Start by storing the extent (20) in the Extent Size lookup table. Also, store the two representations of the house (dot, outline) in their own separate structures. Use the Test Extent OC to test the extent of the house against the lookup table entry containing the size value (20 pixels). Then, use the Conditional Execute structure OC to draw a structure with the dot polymarker representation if the condition flag indicates the size check was less than the table value. In the same manner, use the Conditional Execute structure OC to draw the structure containing the polyline representation of the outline of the house if the size check was greater than or equal to the table value.

PEXOCCApplicationData - Structure Application Data (5.0)

Synopsis

void **PEXOCCApplicationData**(`PEXOCC` *context*, int *length*, char **data*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	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

Invoke **PEXOCCApplicationData** to create an application data output command which has no visible effect when rendered. Use this function 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

PEXOCCConditionFlags - Set Condition Flags (5.2)

Synopsis

void **PEXOCCConditionFlags**(`PEXOCC` *context*, unsigned long *enable*, unsigned long *disable*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
enable	The bits to turn on in the renderer's condition flags.
disable	The bits to turn off in the renderer's condition flags.

Returns

None.

Description

Invoke **PEXOCCConditionFlags** to create an output command that modifies bits in the renderer's condition flag set.

The bits set in the *enable* argument force the corresponding bits in the renderer's condition flags to be set to one. Then, the bits set in the *disable* argument force the corresponding bits in the renderer's condition flags to be set to zero. The implementation applies the *enable* argument first and the *disable* argument last.

Errors

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

See Also

PEXOCCConditionalExecuteStructure, **PEXOCCConditionalIncludeStructure**, **PEXOCCConditionalReturn, PEXOCCTestExtent**, **PEXOCCTestExtent2D**

PEXOCCConditionalExecuteStructure - Conditional Execute Structure (5.2)

Synopsis

void **PEXOCCConditionalExecuteStructure**(`PEXOCC` *context*, unsigned long *mask*, int *test*,
`PEXStructure` *structure*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
mask	The bit mask to use in the conditional test.
test	The type of conditional test to use (**PEXCTAllEnabled, PEXCTAllDisabled, PEXCTNotAllDisabled, PEXCTNotAllEnabled, PEXCTAlwaysPass** or **PEXCTAlwaysFail**).
structure	The resource identifier of the structure [§2.3.19, page 29].

Returns

None.

Description

Invoke **PEXOCCConditionalExecuteStructure** to create a conditional execute structure output command.

Use the *mask* argument to select which bits in the renderer's condition flags are used in the test. The test passes if these selected bits in the renderer's condition flags meet the condition described by the *test* argument. If the test passes, then the implementation performs the same operation as described in "PEXOCCExecuteStructure - Execute Structure (5.0)" on page 428, using the *structure* argument as the structure identifier of the structure to process next. If the test fails, then the implementation continues processing output commands without invoking the structure specified by the *structure* argument.

If *structure* does not exist at the time **PEXOCCConditionalExecuteStructure** is processed, then the implementation issues a *BadPEXOutputCommand* error. You must first create the structure.

Errors

BadPEXOutputCommand	The *structure* argument references an non-existent structure.
BadPEXOutputCommand	The *test* argument is not a registered value.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCConditionFlags, **PEXOCCConditionalIncludeStructure**, **PEXOCCConditionalReturn**, **PEXCreateStructure**, **PEXOCCTestExtent**, **PEXOCCTestExtent2D**

PEXOCCConditionalIncludeStructure - Conditional Include Structure (5.2)

Synopsis

void **PEXOCCConditionalIncludeStructure**(`PEXOCC` *context*, unsigned long *mask*, int *test*,
`PEXStructure` *structure*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
mask	The bit mask to use in the conditional test.
test	The type of conditional test to use (**PEXCTAllEnabled, PEXCTAllDisabled, PEXCTNotAllDisabled, PEXCTNotAllEnabled, PEXCTAlwaysPass** or **PEXCTAlwaysFail**).
structure	The resource identifier of the structure [§2.3.19, page 29].

Returns

None.

Description

Invoke **PEXOCCConditionalIncludeStructure** to create a conditional include structure output command.

Use the *mask* argument to select which bits in the renderer's condition flags are used in the test. The test passes if these selected bits in the renderer's condition flags meet the condition described by the *test* argument. If the test passes, then the implementation performs the same operation as described in "PEXOCCIncludeStructure - Include Structure (5.2)" on page 429, using the *structure* argument as the structure identifier of the structure to process next. If the test fails, then the implementation continues processing output commands without invoking the structure specified by the *structure* argument.

If *structure* does not exist at the time **PEXOCCConditionalIncludeStructure** is processed, then the implementation issues a *BadPEXOutputCommand* error. You must first create the structure.

Errors

BadPEXOutputCommand	The *structure* argument references an non-existent structure.
BadPEXOutputCommand	The *test* argument is not a registered value.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCConditionFlags, **PEXOCCConditionalExecuteStructure**, **PEXOCCConditionalReturn**, **PEXCreateStructure**, **PEXOCCTestExtent**, **PEXOCCTestExtent2D**

PEXOCCConditionalReturn - Conditional Return (5.2)

Synopsis

void **PEXOCCConditionalReturn**(`PEXOCC` *context*, unsigned long *mask*, int *test*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
mask	The bit mask to use in the conditional test.
test	The type of conditional test to use (**PEXCTAllEnabled, PEXCTAllDisabled, PEXCTNotAllDisabled, PEXCTNotAllEnabled, PEXCTAlwaysPass** or **PEXCTAlwaysFail**).

Returns

None.

Description

Invoke **PEXOCCConditionalReturn** to create a conditional return output command.

Use the *mask* argument to select which bits in the renderer's condition flags are used in the test. The test passes if these selected bits in the renderer's condition flags meet the condition described by the *test* argument. If the test passes, then the implementation terminates the processing of output commands in the current structure, restores the saved renderer pipeline state, and then resumes processing output commands in the parent structure. If the implementation is processing the root structure or if this output command is included in a **PEXRenderOutputCommands** or **PEXRenderElements** request, then the implementation ignores all subsequent output commands until encountering the next **PEXEndRendering** or **PEXEndStructure** request.

If the test fails, then the implementation continues processing output commands.

Errors

BadPEXOutputCommand	The *test* argument is not a registered value.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCConditionFlags, **PEXOCCConditionalExecuteStructure**, **PEXOCCConditionalIncludeStructure**, **PEXOCCTestExtent**, **PEXOCCTestExtent2D**

PEXOCCExecuteStructure - Execute Structure (5.0)

Synopsis

void **PEXOCCExecuteStructure**(`PEXOCC` *context*, `PEXStructure` *structure*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
structure	The resource identifier of the structure [§2.3.19, page 29].

Returns

None.

Description

Invoke **PEXOCCExecuteStructure** to create 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 **PEXOCCExecuteStructure** is processed, then the implementation issues a *BadPEXOutputCommand* error. You must first create the structure.

Errors

BadPEXOutputCommand	The *structure* argument references an non-existent structure.
BadPEXRenderer	The specified argument resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

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

PEXOCCIncludeStructure - Include Structure (5.2)

Synopsis

void **PEXOCCIncludeStructure**(`PEXOCC` *context*, `PEXStructure` *structure*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
structure	The resource identifier of the structure [§2.27.3, page 53].

Returns

None.

Description

Invoke **PEXOCCIncludeStructure** to create an include structure output command which causes the specified structure to be included during structure traversal.

The implementation includes the structure by processing all the output commands in the structure without saving or restoring renderer pipeline state or modifying the global or local transforms as part of the include structure processing.

If *structure* does not exist at the time **PEXOCCIncludeStructure** is processed, then the implementation issues a *BadPEXOutputCommand* error. You must first create the structure.

Errors

BadPEXOutputCommand	The *structure* argument references an non-existent structure.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

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

PEXOCCGSE - Generalized Structure Element (5.0)

Synopsis

void **PEXOCCGSE**(`PEXOCC` *context*, long *id*, int *length*, char **data*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	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

Invoke **PEXOCCGSE** to create a generalized structure element.

Each individual PEX implementation should provide you with a complete interface and definition for each GSE identifier. PEXlib does not define any standard PEX GSEs.

If the implementation does not support the specified GSE identifier (*id*), then the implementation ignores this output command.

Errors

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

See Also

PEXOCCLabel - Structure Label (5.0)

Synopsis

void **PEXOCCLabel**(PEXOCC *context*, long *label*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
label	A value to be used as a label.

Returns

None.

Description

Invoke **PEXOCCLabel** to create a label output command which has no visible effect.

The primary use for this function occurs when you store a label in a structure. You can use labels in various element pointer positioning operations. Labels you specify 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

PEXDeleteBetweenLabels, PEXDeleteToLabel, **PEXSetElementPtrAtLabel**

PEXOCCNoop - Noop Output Command (5.1)

Synopsis

void **PEXOCCNoop**(`PEXOCC` *context*)

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

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).

Returns

None.

Description

Invoke **PEXOCCNoop** to create a no-op output command which, when rendered, does 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

PEXOCCSequentialProcessing - Sequential Processing of Output Commands (5.2)

Synopsis

void **PEXOCCSequentialProcessing**(`PEXOCC` *context*, int *seq_proc*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
seq_proc	Flag to control sequential processing (**PEXOn** or **PEXOff**)

Returns

None.

Description

Invoke **PEXOCCSequentialProcessing** to create a sequential processing output command which, when encountered by a renderer, sets the sequential processing state of the renderer.

If *seq_proc* is **PEXOn**, then the renderer guarantees each individual output command for a given request (excluding Execute Structure, Conditional Execute Structure, Include Structure, Conditional Include Structure, GSE, GDP 3D, and GDP 2D output commands) is executed atomically and sequentially.

If *seq_proc* is **PEXOff**, then the renderer is released from the requirement to maintain strict atomic hierarchical order and may schedule parallel processing or reorder processing even though the pixels may arrive in the image buffer in a non-determined order. All attributes and transforms must naturally be correctly bound to the primitives to which they apply.

Errors

BadPEXOutputCommand	The *seq_proc* argument is not **PEXOn** or **PEXOff**.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCTestExtent - Test Extent Output Command (5.2)

Synopsis

void **PEXOCCTestExtent**(`PEXOCC` *context*, `PEXCoord` *corner1*, `PEXCoord` *corner2*, int *index*, int *condition*, unsigned long *clip_test_mask*, unsigned long *size_test_mask*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
corner1	One corner of the extent box [§2.7.1, page 33].
corner2	The other corner of the extent box [§2.7.1, page 33].
index	Index into the renderer's extent size table.
condition	Comparison condition for the extent size (**PEXLessThan, PEXLessThanOrEqual, PEXGreaterThan, PEXGreaterThanOrEqual, PEXEqual** or **PEXNotEqual)**
clip_test_mask	Bitmask indicating bits to be changed in the renderer's condition flags according to the result of the clip test.
size_test_mask	Bitmask indicating bits to be changed in the renderer's condition flags according to the result of the size test.

Returns

None.

Description

Invoke **PEXOCCTestExtent** to create a test extent output command which, when encountered by a renderer, modifies the renderer's pipeline state condition flags based on the results of a clip test and a size test against the given extent.

The values in *corner1* and *corner2* define minimum and maximum *x*, *y*, and *z* coordinates in modeling coordinates. These two corners, as diagonal corners of the extent, define the eight corners of a bounding box. The renderer transforms these eight corners using the current modeling and viewing transformations to obtain the bounding box in NPC space. The renderer does not apply model clipping operations in this transformation. The renderer then modifies the extent to become the smallest box that surrounds itself with the edges being parallel to the NPC axes.

The renderer begins the clip test by clipping the NPC space bounding box and transforming the clipped NPC space bounding box to DC space. The renderer does not apply the clip list or window occlusion during clip testing.

If the extent is completely outside the renderer's viewport or completely outside the renderer's NPC subvolume, then the renderer ORs the *clip_test_mask* bitmask into the renderer's condition flags. Otherwise the renderer ANDs the complement of the *clip_test_mask* bitmask into the renderer's condition flags.

After modifying the condition flags with the results of the clip test, the renderer begins the size test by computing the extent's diagonal length in an implementation-dependent way. Conceptually the renderer uses the Euclidean distance, but may use a different measure, as long as it represents the extent's diagonal size in pixels.

If the extent is completely clipped, then the renderer assumes that the extent size is zero. The renderer again transforms the NPC space bounding box to DC, but without clipping in either NPC or DC, resulting in an extent that represents the full size of the object in DC space, with no clipping applied.

The extent's size is compared to the extent size in the renderer's extent size lookup table entry selected by *index,* using the comparison operation specified by the *condition* argument. If the comparison is true, the renderer ORs the *size_test_mask* bitmask into the renderer's condition flags. Otherwise the renderer ANDs the complement of the *size_test_mask* bitmask into the renderer's condition flags.

The extent tests are defined to err on the side of determining an object is more visible and its size slightly larger than a more precise test would find. Implementations may choose to implement higher quality and more exact extent tests.

Errors

BadPEXOutputCommand	The value of *index* is zero.
BadPEXOutputCommand	*Condition* is not one of the registered relational test values.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCConditionFlags, **PEXOCCConditionalExecuteStructure**, **PEXOCCConditionalIncludeStructure**, **PEXOCCConditionalReturn**, **PEXCreateStructure**, **PEXOCCTestExtent2D**

PEXOCCTestExtent2D - Test Extent 2D Output Command (5.2)

Synopsis

void **PEXOCCTestExtent2D**(`PEXOCC` *context*, `PEXCoord2D` *corner1*, `PEXCoord2D` *corner2*, int *index*, int *condition*, unsigned long *clip_test_mask*, unsigned long *size_test_mask*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
display (OCC)	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id (OCC)	The resource identifier of the renderer or structure.
req_type (OCC)	The request type for the output command (**PEXOCRender**, **PEXOCStore**, **PEXOCRenderSingle** or **PEXOCStoreSingle**).
corner1	One corner of the extent box [§2.7.2, page 33].
corner2	The other corner of the extent box [§2.7.2, page 33].
index	Index into the renderer's extent size table.
condition	Comparison condition for the extent size (**PEXLessThan, PEXLessThanOrEqualTo, PEXGreaterThan, PEXGreaterThanOrEqualTo, PEXEqualTo** or **PEXNotEqualTo)**
clip_test_mask	Bitmask indicating bits to be changed in the renderer's condition flags according to the result of the clip test.
size_test_mask	Bitmask indicating bits to be changed in the renderer's condition flags according to the result of the size test.

Returns

None.

Description

Invoke **PEXOCCTestExtent2D** to create a test extent two-dimensional output command which, when encountered by a renderer, modifies the renderer's pipeline state condition flags based on the results of a clip test and a size test against the given extent.

The command is a 2D form of the Test Extent output command, and differs from it only by containing a 2D value for *corner1* and *corner2*. The z-coordinate of the corners is implicitly zero. Once these implicit values are associated with the output command, it is processed as a Test Extent output command.

Errors

BadPEXOutputCommand	The value of *index* is zero.
BadPEXOutputCommand	*Condition* is not one of the registered relational test values.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXOCCConditionFlags, **PEXOCCConditionalExecuteStructure**, **PEXOCCConditionalIncludeStructure**, **PEXOCCConditionalReturn**, **PEXCreateStructure**, **PEXOCCTestExtent**

Chapter 10

Lookup Tables

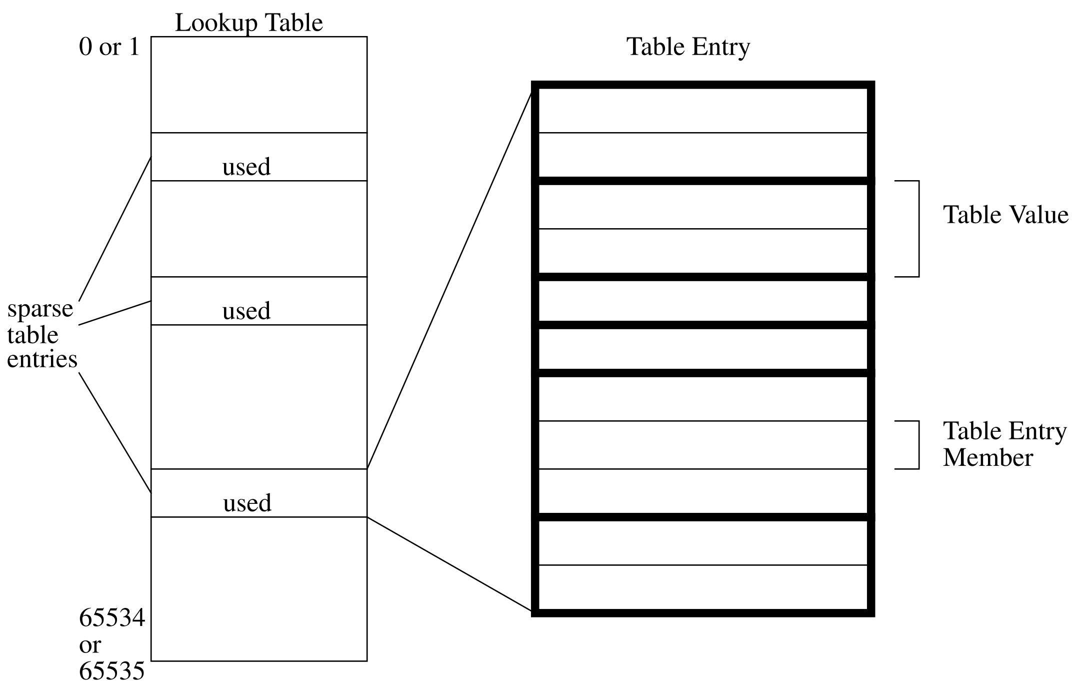
A *lookup table* is a PEX resource that holds collections of related values. When processing output commands, the renderer accesses the data you specify in the lookup tables you define. It may be easier and faster for you to change values in a lookup table, where you group values, rather than editing every single instance of the data you want to change throughout a large structure network. Lookup tables are also used by programs to store information such as light source descriptions, depth-cue parameters, and view definitions.

Lookup tables consist of table entries, described in section 10.1, with each entry holding potentially different values for the set of parameters particular to the table type. Each entry in a table is addressed by an integer index. Depending on the table type, the first index you can define is either 0 or 1 and the maximum index you can define is either 65534 or 65535. Although the minimum and maximum indices theoretically allow for 65543 entries, implementations may offer less. To determine the maximum number of table entries that an implementation supports for each table, invoke the **PEXGetTableInfo** function (see “PEXGetTableInfo - Get Lookup Table Information (5.0)” on page 481.)

You do not have to define sequential indices for entries in a table. Your table could contain entries 5, 7, and 10, for example, without containing entries 1 through 4, 6, 8, or 9. If a table has no entry for a particular index, then that entry is said to be undefined for that table. Initially, all entries not predefined are undefined.

You must explicitly create each table by invoking the **PEXCreateLookupTable** function (see “PEXCreateLookupTable - Create Lookup Table (5.0)” on page 476). Add or redefine entries by invoking the **PEXChangeTableValues** function (see “PEXChangeTableValues - Change Table Values (5.2)” on page 474). Delete entries with the **PEXDeleteTableEntries** function (“PEXDeleteTableEntries - Delete Lookup Table Entries (5.0)” on page 477) when they are no longer needed.

Conceptual Diagram of a PEX Lookup Table



10.0.1. Fallback Entries and Values

Each lookup table has a designated fallback entry; a default value if no table entries are defined. If the renderer tries to access an undefined table entry during rendering, then the renderer uses the fallback table entry instead. If the fallback entry is undefined, then the renderer uses the fallback table values defined for each field of the table. The fallback entry and fallback values are listed with the definition of each table type in Section 10.1.

If the renderer attempts to use a lookup table which has a renderer attribute value of Null, then the results are as defined in "Renderers" on page 507. The renderer uses the fallback values if your application references a valid table resource but the selected and fallback entries are undefined.

10.0.2. Table Values and Table Entries

Prior to PEXlib 5.2, you were able to edit lookup table entries only by replacing the entire table entry. Beginning with PEXlib 5.2, the table entry manipulation functions (**PEXChangeTableValues** and **PEXGetTableValues**) enable you to alter or retrieve portions of a table entry, instead of the entire entry. These table entry portions are called "values" and may consist of one or several members of the table values data structure. To indicate which portions of the table entry you intend to edit or retrieve, use the bitmask argument included with these functions. When using the **PEXChangeTableValues** and **PEXGetTableValues** functions, always use the `PEX<table name>Values` data structures to access the data.

The older table entry manipulation functions (**PEXSetTableEntries**, **PEXGetTableEntry**, and **PEXGetTableEntries**) are still supported for the pre-PEXlib 5.2 table types, but because only the newer functions enable you access to all of the table types, you should avoid using these older table entry manipulation functions in newer applications. Also, some pre-PEXlib 5.2 table types now offer new members, which you can access only with the newer functions. When using these older functions, always use the `PEX<table name>Entry` data structures.

10.0.3. Predefined Entries

A PEX implementation may predefine some entries in a lookup table when the application creates the table. These are entries that the implementation fills in automatically. Predefined entries are sequential and include the first legal entry of the table. The number of entries predefined is implementation-dependent, may be zero, and may vary for each table type. The client can redefine predefined entries. To determine the number of predefined entries for a particular table type, invoke the **PEXGetTableInfo** function (see "PEXGetTableInfo - Get Lookup Table Information (5.0)" on page 481.) To determine the actual entries that a implementation predefines for a particular table type and the sample drawable, invoke the **PEXGetPredefinedEntries** function (see "PEXGetPredefinedEntries - Get Lookup Table Pre-defined Indices (5.0)" on page 485.)

As of PEXlib 5.2, it is recommended that implementations *not* provide predefined entries because of the large variation in the predefined values found in different implementations. Therefore, do not rely on predefined entries when writing PEXlib programs. The concept is explained here and supported to maintain compatibility with PEX versions previous to 5.2.

10.0.4. Set and Realized Values

If you set a particular value in a table entry, it is possible that the implementation can not represent the exact value you specify. Therefore, your implementation may round the value to a lower precision, or modify the value somewhat to the closest value it can handle. In these cases, the implementation silently maps the value to an approximation. This mapped value is called the "realized" value. To determine the realized value (the value your implementation actually uses when rendering) invoke the **PEXGetTableValues** function (see "PEXGetTableValues - Get Table Values (5.2)" on page 482.)

10.0.5. Resources in Table Entries

Some table entry types have members that store references to other resources. If you would like to modify a table entry member to indicate that no resource is to be used, then store the value **None** in the member.

10.1. Table Types, Availability, and Definitions

The following section describes each table in terms of its table entry, the table semantics, and the PEXlib data structure used to manipulate the table entries. Some of the functions described in the following section require a *table type* argument. The complete list of table types and their corresponding table entry data structure names are:

Table 11: Table Types and Data Structures

Table Type	Table Value Data Structure	PEX
PEXLUTColor	PEXColorValues [§2.16.2, page 39]	5.0
PEXLUTColorApprox	PEXColorApproxValues [§2.16.4, page 40]	5.0
PEXLUTDataMapping	PEXDataMappingValues [§2.16.5, page 40]	5.2
PEXLUTDepthCue	PEXDepthCueValues [§2.16.7, page 40]	5.0
PEXLUTEdgeBundle	PEXEdgeBundleValues [§2.16.9, page 41]	5.0
PEXLUTExtentSize	PEXExtentSizeValues [§2.16.10, page 41]	5.2
PEXLUTHighlight	PEXHighlightValues [§2.16.11, page 41]	5.2
PEXLUTInteriorBundle	PEXInteriorBundleValues [§2.16.13, page 42]	5.0
PEXLUTLight	PEXLightValues [§2.16.15, page 42]	5.0
PEXLUTLineBundle	PEXLineBundleValues [§2.16.17, page 43]	5.0
PEXLUTMarkerBundle	PEXMarkerBundleValues [§2.16.19, page 43]	5.0
PEXLUTMarkerGlyph	PEXMarkerGlyphValues [§2.16.20, page 43]	5.2
PEXLUTPattern	PEXPatternValues [§2.16.22, page 43]	5.0
PEXLUTTextBundle	PEXTextBundleValues [§2.16.25, page 44]	5.0
PEXLUTTextFont	PEXTextFontValues [§2.16.27, page 44]	5.0
PEXLUTTextureMapping	PEXTMValues [§2.16.28, page 44]	5.2
PEXLUTView	PEXViewValues [§2.16.30, page 45]	5.0

The table entries are described in terms of table values, where one or several table items (structure members) are grouped under a table value. For example, matrices which appear in table entries are grouped with their corresponding matrix hints because they logically depend on each other and need to be changed simultaneously by the application. The name of each table value is the PEXlib symbol (beginning with **PEXLUTV**) that you should use when building the bitmask needed for the table entry manipulation functions **PEXChangeTableValues** and **PEXGetTableValues**. If you intend to change or get an entire table entry (all values), there is a convenience symbol for setting all the bits.

For table types where some of the values have corresponding attribute output commands, the Description column in the tables on the following pages includes a cross-reference to the page for that output command. Please see the specification of the output command for more information about the behavior and allowed values for the corresponding table value. Otherwise, the table value is explained in the text following the table.

To get or set the values you are interested in, use the PEXlib table value data structure. You only need to get or set the members of the structure that correspond to table entry values that you have indicated interest in through the use of the bitmask.

Note: Applications should always use the PEX<table name>Values data structures when manipulating table values with the **PEXChangeTableValues** and **PEXGetTableValues** functions. Do not use the PEX<table name>Entry data structures. You should only use the PEX<table name>Entry data structures with the old table entry manipulation functions described in "Lookup Table Requests for PEXlib 5.0 and 5.1 Compatibility" on page 483.

10.1.1. Color Table

Use color lookup tables to define how the implementation resolves indirect color references during rendering.

Table Value Mask	Description	Fallback	PEX
PEXLUTVColorColor			5.0
	color type	**PEXColorTypeRGB**	
	unused		
	color value	(1.0, 1.0, 1.0)	
PEXLUTVColorAll5_1 (selects entire table entry)			5.0
PEXLUTVColorAll5_2 (selects entire table entry)			5.0

Color table entries hold a single direct-color value and the color type of that value. The color type need not be the same among color table entries. You can store only direct colors in color lookup tables; it is not possible to store a color of type **PEXColorTypeIndexed**.

The implementation accesses the color table during rendering whenever it encounters a color of type **PEXColorTypeIndexed**. The value you specify for the **PEXColorTypeIndexed** color indicates the entry of the color table.

PEX treats direct color components that are out of range in an implementation-dependent way. If you specify an unsupported color type or a color type of **PEXColorTypeIndexed**, then the implementation issues a *BadPEXColorType* error.

Valid color table indices are in the range [0, 65534]. The fallback entry is entry 1.

10.1.2. Color Approximation Table

Color approximation lookup tables hold the information necessary to convert PEX colors to X11 colormap pixel values.

Table Value Mask	Description	Fallback	PEX
PEXLUTVColorApproxMethod			5.0
	color approximation type	(imp-dep)	
PEXLUTVColorApproxModel			5.0
	color approximation model	(imp-dep)	
PEXLUTVColorApproxMax1			5.0
	max1	(imp-dep)	
PEXLUTVColorApproxMax2			5.0
	max2	(imp-dep)	
PEXLUTVColorApproxMax3			5.0
	max3	(imp-dep)	
PEXLUTVColorApproxDither			5.0
	dither	(imp-dep)	
PEXLUTVColorApproxMult1			5.0
	mult1	(imp-dep)	
PEXLUTVColorApproxMult2			5.0
	mult2	(imp-dep)	
PEXLUTVColorApproxMult3			5.0
	mult3	(imp-dep)	
PEXLUTVColorApproxWeight1			5.0
	weight1	(imp-dep)	
PEXLUTVColorApproxWeight2			5.0
	weight2	(imp-dep)	
PEXLUTVColorApproxWeight3			5.0
	weight3	(imp-dep)	
PEXLUTVColorApproxBasePixel			5.0
	base pixel	(imp-dep)	
PEXLUTVColorApproxAll5_1 (selects entire table entry)			5.0
PEXLUTVColorApproxAll5_2 (selects entire table entry)			5.0

Color approximation converts colors computed by the PEX rendering pipeline to pixel values. The PEX rendering pipeline produces a color in a multi-dimensional color space such as RGB. The PEX color approximation process converts that color, called the *input color*, to a pixel value that selects an entry in an X11 colormap. The method and arguments used to convert the color are taken from the color approximation table entry indicated by the pipeline state's *color approximation index*. Alpha values are unaffected by color approximation.

PEXlib defines the following values:

color approximation type: The color approximation type specifies the method and parameters which are used to convert the input color to a pixel value that selects an entry in an X11 colormap.

PEXColorSpace - intended for use with *DirectColor*, *TrueColor*, *PseudoColor*, and *StaticColor* visuals.

PEXColorRange - intended for use with *GrayScale* and *StaticGray* visuals.

color approximation model: During the color approximation process, each color produced by the rendering pipeline is first converted to the color model specified in the selected color approximation table entry. (When that color model is the same as that of the input color, this is an identity mapping.)

PEXColorApproxRGB - specifies a red, green, blue color model

PEXColorApproxCIE - specifies a CIELUV color model

PEXColorApproxHSV - specifies hue, saturation, value

PEXColorApproxHLS - specifies hue, lightness, saturation

PEXColorApproxYIQ - specifies NTSC luminance (Y), inphase (wideband orange-cyan), quadrature (narrowband magenta-green)

After the input color is converted to the color approximation color model, the converted color is converted to a pixel value according to the method you indicate by the *color approximation type* in the selected table entry.

ColorSpace Color Approximation

max1, max2, max3: In *ColorSpace* color approximation, PEX scales each component of a color, (C_1, C_2, C_3) by the corresponding *max* value, (*max1, max2, max3*), in the selected color approximation table entry, as follows:

$$C'_1 = max1 \cdot C_1, C'_2 = max2 \cdot C_2, C'_3 = max3 \cdot C_3$$

The C_i' are conceptually rounded to integers in this operation.

max1, *max2*, and *max3* typically indicate the number of samplings (minus 1) of each color component in a colormap organized as a color cube. For example, the *max* values in a colormap organized as a 3-3-2 color cube are 7, 7, 3 (2^3-1, 2^3-1, 2^2-1). Thus, the equation serves to scale the color coordinates into the available range of discrete colors.

The C_i' are converted to a pixel value by the formula given in the following equation:[1]

$$P = C'_1 \cdot mult1 + C'_2 \cdot mult2 + C'_3 \cdot mult3 + basePixel$$

mult1, *mult2*, *mult3*, and *base pixel:* In *ColorSpace* color approximation, the implementation combines the scaled color components to produce a pixel value *P* which is written to the drawable. Typically you use *mult1, mult2*, and *mult3* to shift the color components to the bits within the pixel representing that color component. For example, the *mult* values in a colormap organized as a 3-3-2 color cube, where bits 5 - 7 of the pixel represent the C_1 component, bits 2 - 4 the C_2 component, and bits 0 and 1 the C_3 component, would be 2^5, 2^2, and 1. Use the *base pixel* to shift the values up to the first pixel of the color cube within the colormap.

ColorSpace color approximation does not use the *weight1*, *weight2*, and *weight3* values in the color approximation entry.

ColorRange Color Approximation

weight1, weight2, and weight3: In *ColorRange* color approximation, PEXlib multiplies the color components in the color approximation color model, (C_1, C_2, C_3), by the corresponding weights, (*weight1*, *weight2*, *weight3*), taken from

1. The use of addition rather than logical OR for composing pixel values permits allocations where the primary components are not allocated into distinct bitplanes.

the color approximation entry. These terms are added together to form a single, conceptual floating-point value:

$$V = C_1 \cdot weight1 + C_2 \cdot weight2 + C_3 \cdot weight3$$

The weights may emphasize, minimize, or eliminate one or more of the color components. For example, weights of (.299R,.587G,.114B) convert an RGB value to a single-valued intensity used in the NTSC color standard. Typically, you should normalize the weights so that they sum to 1; PEX does not normalize the weights.

After the weights are applied, the computed value, *V*, is multiplied by *max1* in the color approximation entry:

$$V' = V \cdot max1$$

max2 and *max3* are not used.

The value *V'* is replicated for each of the three color components. Each component is then multiplied by *mult1*, *mult2*, or *mult3*, respectively, and the values are added together with *base pixel* to form the pixel value, *P*, written to the drawable:[1]

$$P = V' \cdot mult1 + V' \cdot mult2 + V' \cdot mult3 + basePixel$$

max1 is typically one less than the number of entries in a colormap. For example, a value of 99 for *max1* scales colors to the range [0, 99], and might be used to display an image in gray scale using a colormap with 100 contiguous shades of gray. Typically *mult1* is 1 and *mult2* and *mult3* are 0. PEX supports *mult2* and *mult3* so you can simulate gray ramps on a *DirectColor* visual by replicating the intensity in all three channels of the pixel.

Multipliers

Inquire *BestColorApproxValues* with request **PEXGetImpDepConstants** to determine what values, if any, of *mult1, mult2,* and *mult3* are likely to result in the best performance.

Dithering

dither is a hint to the renderer as to whether some attempt at dithering (manipulation of pixel values in order to smooth the image) should be performed. With request **PEXGetImpDepConstants** inquire *DitheringSupported* to determine if the implementation can dither, and, if the implementation does support dithering, inquire *DitheringControllable* to determine if you can disable dithering with the dithering hint. Though the implementation might support dithering, the algorithm used is implementation-dependent.

The color approximation table is always accessed when rendering. The value of the pipeline state's *color approximation index* selects the desired entry of the table.

If you specify an unsupported color approximation type or unsupported color approximation model, then the implementation issues a *BadValue* error.

Valid color approximation lookup table indices are in the range [0, 65534]. The fallback entry is entry 0.

1. The use of addition rather than logical OR for composing pixel values permits allocations where the primary components are not allocated into distinct bitplanes.

10.1.3. Data Mapping Table

Use data mapping lookup tables to define how the implementation maps color or other application-specific data in order to determine the intrinsic color of fill area output primitives with extended data.

Table Value Mask	Description	Fallback	PEX
PEXLUTVDataMappingMethod			5.2
	data mapping method (page 308)	**PEXDMMNone**	
PEXLUTVDataMappingSourceSelectors			5.2
	number of source selectors	3	
	prioritized list of source selectors	{**PEXVertexColor**, **PEXFacetColor**, **PEXSurfaceColor**}	
PEXLUTVDataMappingAll5_2 (selects entire table entry)			5.2

Data mapping is the first stage in the color pipeline.

The superseded functions **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on data mapping lookup tables.

PEXlib defines the following values:

data mapping method:

> **PEXDMMNone** - indicates no data mapping method.
>
> **PEXDMMColor** - maps color, or other application-specific data, to intrinsic color.

The data mapping table is accessed during rendering whenever the pipeline state's *data map method ASF* or *bf data map method ASF* entry is **PEXBundled**. The values of the pipeline state's *data map index* and *bf data map index* select the desired entries of the table.

If you specify an unsupported data mapping method, if the data record size doesn't match the method, or if the method is **PEXDMMColor** and you specify an undefined source selector, then the implementation issues a *BadValue* error.

Valid data mapping table indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.4. Depth Cue Table

Use depth cueing lookup tables to blend color computed by the lighting, shading, and texturing stages of the rendering pipeline with a depth cue color. The blending is controlled relative to the depth in NPC space. Use depth cueing, for example, to blend the portion of the image furthest away from the viewer with the background color, heightening the sensation of depth in the image. Color alpha components are not depth cued.

Table Value Mask	Description	Fallback	PEX
PEXLUTVDepthCueMode			5.0
	depth cue mode	**PEXOff**	
PEXLUTVDepthCueFrontReferencePlane			5.0
	front reference plane	1.0	
PEXLUTVDepthCueBackReferencePlane			5.0
	back reference plane	0.0	
PEXLUTVDepthCueFrontScaleFactor			5.0
	front scale factor	1.0	
PEXLUTVDepthCueBackScaleFactor			5.0
	back scale factor	0.0	
PEXLUTVDepthCueColor			5.0
	depth cue color	(**PEXColorTypeRGB**, (0.0, 0.0, 0.0))	
PEXLUTVDepthCueAll5_1 (selects entire table entry)			5.0
PEXLUTVDepthCueAll5_2 (selects entire table entry)			5.0

PEXlib defines the following values:

depth cue mode:

PEXOff - does not perform depth-cueing.

PEXOn - performs depth-cueing.

depth cue color: Primitives are depth cued by blending the reflected and textured color of their points with the specified *depth cue color*.

reference planes: The front and back reference planes are perpendicular to the NPC z axis, and are specified by the Z_{NPC} coordinate at which they intersect that axis.

scale factors: The scale factors must be in the range [0.0, 1.0].

The depth cueing table is always accessed when rendering. The value of the pipeline state's *depth cueing index* selects the desired entry of the table.

Direct color components that are out of range are treated in an implementation-dependent way. If you specify an unsupported color type, then the implementation issues a *BadPEXColorType* error. If the color type is **PEXColorTypeIndexed** and the color index is 65535, then the implementation issues a *BadValue* error. If you specify an undefined depth cue mode, if the *back reference plane* is greater than the *front reference plane*, or if the *front scale factor* or *back scale factor* is not in the range [0.0, 1.0], then the implementation issues a *BadValue* error.

Valid depth cue table indices are in the range [0, 65534]. The fallback entry is entry 0.

10.1.5. Edge Bundle Table

Edge bundle lookup tables contain collections of edge attribute values.

Table Value Mask	Description	Fallback	PEX
PEXLUTVEdgeBundleFlag			5.0
	edge flag (page 395)	**PEXOff**	
PEXLUTVEdgeBundleType			5.0
	edge type (page 396)	**PEXLineTypeSolid**	
PEXLUTVEdgeBundleWidth			5.0
	edge width (page 398)	1.0	
PEXLUTVEdgeBundleColor			5.0
	edge color (page 393)	(**PEXColorTypeIndexed**, 1)	
PEXLUTVEdgeBundleAll5_1 (selects entire table entry)			5.0
PEXLUTVEdgeBundleAll5_2 (selects entire table entry)			5.0

The edge bundle table is accessed during rendering whenever the aspect source flag of one of the edge attributes stored in the edge bundle table is **PEXBundled**. The aspect source flag for each attribute is a member of the pipeline state. The value of the pipeline state's *edge bundle index* selects the desired entry of the table. The value of the bundled edge attributes in that entry is then used to set the values of those edge attributes in the pipeline state.

Direct color components that are out of range are treated in an implementation-dependent way. If you specify an unsupported color type, then the implementation issues a *BadPEXColorType* error. If the color type is **PEXIndexed** and the color index is 65535, then the implementation issues a *BadValue* error. If the surface edge flag is not **PEXOn** or **PEXOff**, or if you specify an unsupported edge type, then the implementation issues a *BadValue* error.

Valid edge bundle lookup table indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.6. Extent Size Table

Extent size lookup tables hold the extent sizes used with **PEXTestExtent** and **PEXTestExtent2D** OC's (page 434) and (page 436). These output commands compare the length of the diagonal of an extent volume or rectangle with an extent size obtained from the table. The results of this comparison determine modifications to the pipeline state's *condition flags* which are used to control the behavior of subsequent **PEXConditionalExecuteStructure** (page 425), **PEXConditionalIncludeStructure** (page 426), and **PEXConditionalReturn** (page 427) OC's.

Table Value Mask	Description	Fallback	PEX
PEXLUTVExtentSizeSize			5.2
	extent size (DC's)	0	
PEXLUTVExtentSizeAll5_2 (selects entire table entry)			5.2

The superseded functions **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on extent size lookup tables.

The implementation accesses the extent size table during rendering whenever it encounters a **PEXTestExtent** or **PEXTestExtent2D** output command. The value of the output command's *index* argument selects the desired entry of the table.

Valid extent size table indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.7. Highlight Table

Highlight table entries indicate how primitives are highlighted.

Table Value Mask	Description	Fallback	PEX
PEXLUTVHighlightMethod			5.2
	highlight method	**PEXHighlightImpDep**	
PEXLUTVHighlightColor			5.2
	highlight color	(imp-dep)	
PEXLUTVHighlightAll5_2 (selects all table values)			5.2

The superseded requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on highlight lookup tables.

PEXlib defines the following values:

highlight method: Each method indicates how the specified color is used.

> **PEXHighlightImpDep** - highlights a primitive in an implementation-dependent fashion.
>
> **PEXHighlightColor** - highlights a primitive by overriding its color with the color specified by the *highlight color* table value of the active highlight color table entry. The implementation does *not* apply lighting, shading, depth cueing, transparency, texture mapping, nor data mapping to the primitive. The implementation applies the color to both the front-facing and back-facing facets of surface primitives.

Remove highlighting by clearing and re-rendering the image.

highlight color: specifies the color used. The resulting color overrides the primitive's color for highlight methods that use color.

Highlighting is performed when a primitive is rendered and the pipeline state's *name set* is contained in the renderer's *highlight inclusion* name set and not in the renderer's *highlight exclusion* name set. The value of the pipeline state's *highlight index* selects the desired entry of the table.

Direct color components that are out of range are treated in an implementation-dependent way. If you specify an unsupported color type, then the implementation issues a *BadPEXColorType* error. If the color type is **PEXColorTypeIndexed** and the color index is 65535, then the implementation issues a *BadValue* error. If you specify an unsupported highlight method, then the implementation issues a *BadValue* error.

Valid highlight table indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.8. Interior Bundle Table

Interior bundle lookup tables hold collections of interior attribute values.

Table Value Mask	Description	Fallback	PEX
PEXLUTVInteriorBundleStyle			5.0
	interior style (page 331)	**PEXInteriorStyleHollow**	
PEXLUTVInteriorBundleIndex			5.0
	interior style index (page 333)	1	
PEXLUTVInteriorBundleReflectionModel			5.0
	reflection model (page 381)	**PEXRMNone**	
PEXLUTVInteriorBundleInterpMethod			5.0
	surface interpolation method (page 399)	**PEXSurfaceInterpNone**	
PEXLUTVInteriorBundleBFStyle			5.0
	back-facing interior style (page 283)	**PEXInteriorStyleHollow**	
PEXLUTVInteriorBundleBFIndex			5.0
	back-facing interior style index (page 285)	1	
PEXLUTVInteriorBundleBFReflectionModel			5.0
	back-facing reflection model (page 290)	**PEXRMNone**	
PEXLUTVInteriorBundleBFInterpMethod			5.0
	back-facing surface interpolation method (page 297)	**PEXSurfaceInterpNone**	
PEXLUTVInteriorBundleSurfaceApprox			5.0
	surface approximation (page 387)	(1 (imp-dep), n/a, n/a)	
PEXLUTVInteriorBundleSurfaceColor			5.0
	surface color (page 390)	(**PEXColorTypeIndexed**, 1)	
PEXLUTVInteriorBundleReflectionAttributes			5.0
	reflection attributes (page 379)	(1.0, 1.0, 1.0, 0.0, 0,0, (**PEXIndexed**, 1))	
PEXLUTVInteriorBundleBFSurfaceColor			5.0
	back-facing surface color (page 294))	(**PEXColorTypeIndexed**, 1)	
PEXLUTVInteriorBundleBFReflectionAttributes			5.0
	back-facing reflection attributes (page 288)	(1.0, 1.0, 1.0, 0.0, 0.0, (**PEXIndexed**, 1))	
PEXLUTVInteriorBundleParaSurfCharacteristics[a]			5.2
	parametric surface characteristics (page 364)	(**PEXPSCNone**, n/a)	
PEXLUTVInteriorBundleAll5_1 (selects entire table entry except 5.2 items)			5.0
PEXLUTVInteriorBundleAll5_2 (selects entire table entry)			5.2

a. The requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on the parametric surface characteristics value of an interior table entry.

If the renderer's *PHIGS interior bundle mode* is **False**, then the implementation accesses the interior bundle table during rendering whenever the aspect source flag of one of the interior attributes stored in the interior bundle table is

PEXBundled. The aspect source flag for each attribute is a member of the pipeline state. The value of the pipeline state's *interior bundle index* selects the desired entry of the table. The implementation ignores the pipeline state's *back-facing interior bundle index, reflectance bundle index, back-facing reflectance bundle index,* and *parametric surface bundle index*: the value of the bundled interior attributes in that entry is used to set the values of these interior attributes in the pipeline state.

When the renderer's *PHIGS interior bundle mode* is **True**, the implementation still accesses the interior bundle during rendering whenever the aspect source flag of one of the interior attributes stored in the interior bundle table is **PEXBundled**. However, which pipeline state index is used to select the desired entry of the table depends on which interior attribute is bundled. Table 12, "Indices Used to Select Bundled Interior Attributes in the PHIGS Interior Bundle Mode," on page 451 defines which pipeline index is used for each interior attribute in the Interior Bundle Table.

Table 12: Indices Used to Select Bundled Interior Attributes in the PHIGS Interior Bundle Mode

Indices	Bundled Interior Attributes
interior bundle index	Surface Interior Style (page 331)
	Surface Interior Style Index (page 333)
	Surface Color (page 390)
	Surface Interpolation Method (page 399)
back-facing interior bundle index	Back-facing Surface Interior Style (page 283)
	Back-facing Surface Interior Style Index (page 285)
	Back-facing Surface Color (page 390)
	Back-facing Surface Interpolation Method (page 297)
reflectance bundle index	Surface Reflection Model (page 381)
	Surface Reflection Attributes (page 379)
back-facing reflectance bundle index	Back-facing Surface Reflection Model (page 290)
	Back-facing Surface Reflection Attributes (page 288)
parametric surface bundle index	Surface Approximation (page 309)
	Parametric Surface Characteristics (page 364)

For more information about each surface interior attribute, see the corresponding surface interior attribute output command. (These output commands set pipeline state which is used whenever the aspect source flag for a surface interior attribute is **PEXIndividual** rather than **PEXBundled**.)

Direct color components that are out of range are treated in an implementation-dependent way. If you specify an unsupported color type, then the implementation issues a *BadPEXColorType* error. If the color type is **PEXColorTypeIndexed** and the color index is 65535, then the implementation issues a *BadValue* error. If you specify an unsupported *interior style*, *back-facing interior style*, *reflection model*, *back-facing reflection model*, *surface interpolation method*, *back-facing surface interpolation method*, *surface approximation method*, or *parametric surface characteristics type*, if a *chordal size* or *chordal deviation surface approximation method tolerance* is non-positive, if a relative *surface approximation method tolerance* is outside of the range [0.0, 1.0], if the *parametric surface characteristics data record size* doesn't match the type, if you specify an *isoparametric curve parametric surface characterization* placement value that is unsupported, or if a *level curves parametric surface characterization direction vector* has a zero length, then the implementation issues a *BadValue* error.

Valid interior bundle lookup table indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.9. Light Table

Each entry of a light lookup table holds a light source definition.

Table Value Mask	Description	Fallback	PEX
PEXLUTVLightType			5.0
	light type	(imp-dep)	
PEXLUTVLightDirection			5.0
	direction vector	(imp-dep)	
PEXLUTVLightPosition			5.0
	position	(imp-dep)	
PEXLUTVLightConcentration			5.0
	concentration	(imp-dep)	
PEXLUTVLightSpreadAngle			5.0
	spread angle	(imp-dep)	
PEXLUTVLightConstantAttenFactor			5.0
	constant attenuation factor	(imp-dep)	
PEXLUTVLightLinearAttenFactor			5.0
	linear attenuation factor	(imp-dep)	
PEXLUTVLightColor			5.0
	light color	(imp-dep)	
PEXLUTVLightConcentrationClampRegion[a]			5.2
	concentration clamp region	(imp-dep)	
PEXLUTVLightQuadraticAttenFactor[a]			5.2
	quadratic attenuation factor	(imp-dep)	
PEXLUTVLightAll5_1 (selects entire table entry except 5.2 items)			5.0
PEXLUTVLightAll5_2 (selects entire table entry)			5.2

a. The requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on the concentration clamp region or quadratic attenuation factor values of a light table entry.

PEXlib defines the following values:

light type:

PEXLightAmbient - illuminates primitives uniformly independent of location and orientation.

PEXLightWCVector - simulates light emanating from infinity along a particular direction.

PEXLightWCPoint - simulates light emanating in all directions from a particular position. Control is provided over the rate at which the intensity drops off as a function of the distance from the light source.

PEXLightWCSpot - simulates a spot light. This light emanates from a particular position within a cone of influence centered along a particular direction. Control is provided over the rate at which the intensity drops off both as a function of the distance from the light source and the distance from the center of the cone of influence. Additional control is provided over how sharply the intensity drops off along a region against the edge of the cone of influence.

Not every light type requires that you define all of the light table fields. The implementation ignores fields that are not needed for the definition of a particular light.

direction: The vector (in WCs) along which the light travels from its source. Applies to the definition of light types **PEXLightWCVector** and **PEXLightWCSpot**.

position: The location (in WCs) from which the light emanates. Applies to the definition of light types **PEXLight-WCPoint** and **PEXLightWCSpot**.

concentration: The factor used to define the rate at which a light's intensity drops off as a function of the distance from the center of the cone of influence to the edge of the cone of influence. Applies to the definition of light type **PEXLight-WCSpot**.

spread_angle: The angle (in radians) from the light source direction vector which defines the cone of influence of the light. The light has no affect on points which lie outside the cone of influence. Applies to the definition of light type **PEXLightWCSpot**.

constant_atten_factor: The constant factor used in determining the rate at which the light's intensity falls off as a function of the distance from the light's position. Applies to the definition of light types **PEXLightWCPoint** and **PEX-LightWCSpot**.

linear_atten_factor: The linear factor used in determining the rate at which the light's intensity falls off as a function of the distance from the light's position. Applies to the definition of light types **PEXLightWCPoint** and **PEXLightWC-Spot**.

light_color: The light's color. Applies to the definition of all types of lights.

concentration_clamp_region: The angle (in radians) inward from the cone of influence edge which defines a region along the outer portion of the cone influence. In this region the concentration is multiplied by a linear ramp which drops from 1.0 at the edge of the region to 0.0 at the edge of the cone of influence. This has the affect of clamping the light's intensity sharply down to zero at the edge of the cone of influence.
Inquire *ConcentrationClampSupported* with request **PEXGetImpDepConstants** to determine whether the concentration clamp region is actually applied in the reflectance calculation.
Applies to the definition of light type **PEXLightWCSpot**.

quadratic_atten_factor: The quadratic factor used in determining the rate at which the light's intensity falls off as a function of the distance from the light's position.
Inquire *QuadraticAttenSupported* with request **PEXGetImpDepConstants** to determine whether the quadratic attenuation factor is actually used in the reflectance calculation.
Applies to the definition of light types **PEXLightWCPoint** and **PEXLightWCSpot**.

The reflectance equations are implementation-dependent.

The implementation accesses the light table during rendering whenever the pipeline state's *light source state* is non-empty. The *light source state* selects entries of the light table which then represent the enabled lights.

Direct color components that are out of range are treated in an implementation-dependent way. If you specify an unsupported color type, then the implementation issues a *BadPEXColorType* error. If the color type is **PEXColorTypeIndexed** and the color index is 65535, then the implementation issues a *BadValue* error. If you specify an unsupported light type or specify a direction vector of length zero, then the implementation issues a *BadValue* error.

Valid light table indices are in the range [1, 65535]. There is not a fallback entry for the light table.

10.1.10. Line Bundle Table

Line bundle lookup tables contain collections of line attribute values.

Table Value Mask	Description	Fallback	PEX
PEXLUTVLineBundleType			5.0
	line type (page 345)	**PEXLineTypeSolid**	
PEXLUTVLineBundleInterpMethod			5.0
	interpolation method (page 374)	**PEXPolylineInterpNone**	
PEXLUTVLineBundleCurveApprox			5.0
	curve approximation (page 305)	(1 (imp-dep), n/a)	
PEXLUTVLineBundleWidth			5.0
	line width scale factor (page 347)	1.0	
PEXLUTVLineBundleColor			5.0
	line color (page 339)	(**PEXColorTypeIndexed**, 1)	
PEXLUTVLineBundleCapStyle[a]			5.2
	wide line cap style (page 338)	**PEXCapStyleButt**	
PEXLUTVLineBundleJoinStyle[a]			5.2
	wide line join style (page 342)	**PEXJoinStyleImpDep**	
PEXLUTVLineBundleAll5_1 (selects entire table entry except 5.2 items)			5.0
PEXLUTVLineBundleAll5_2 (selects entire table entry)			5.2

a. The requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on the line cap style or join style values of a line bundle entry.

The implementation accesses the line bundle table during rendering whenever the aspect source flag of one of the line attributes stored in the line bundle table is **PEXBundled**. The aspect source flag for each attribute is a member of the pipeline state. The value of the pipeline state's *line bundle index* selects the desired entry of the table. The value of the bundled line attributes in that entry is then used to set the values of those line attributes in the pipeline state.

For more information about each line attribute, see the corresponding line attribute output command. (These output commands set pipeline state which is used whenever the aspect source flag for a line attribute is **PEXIndividual** rather than **PEXBundled**.)

Direct color components that are out of range are treated in an implementation-dependent way. If you specify a color type that is unsupported, then the implementation issues a *PEXBadColorType* error. If the color type is **PEXColorTypeIndexed** and the color index is 65535, then the implementation issues a *BadValue* error. If you specify an unsupported *line type*, *interpolation method*, *curve approximation method*, *cap style*, or *join style*, if a *chordal size* or *chordal deviation curve approximation method tolerance* is non-positive, or if a relative *curve approximation method tolerance* is outside of the range [0.0, 1.0], then the implementation issues a *BadValue* error.

Valid line bundle indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.11. Marker Bundle Table

Marker bundle lookup tables contain collections of marker attribute values.

Table Value Mask	Description	Fallback	PEX
PEXLUTVMarkerBundleType			5.0
	marker type (page 357)	**PEXMarkerAsterisk**	
PEXLUTVMarkerBundleScale			5.0
	marker scale factor (page 356)	1.0	
PEXLUTVMarkerBundleColor			5.0
	marker color (page 353)	(**PEXColorTypeIndexed**, 1)	
PEXLUTVMarkerBundleAll5_1 (selects entire table entry)			5.0
PEXLUTVMarkerBundleAll5_2 (selects entire table entry)			5.0

The implementation accesses the marker bundle table during rendering whenever the aspect source flag of one of the marker attributes stored in the marker bundle table is **PEXBundled**. The aspect source flag for each attribute is a member of the pipeline state. The value of the pipeline state's *marker bundle index* selects the desired entry of the table. The value of the bundled marker attributes in that entry is then used to set the values of those marker attributes in the pipeline state.

For more information about each marker attribute, see the corresponding marker attribute output command. (These output commands set the pipeline state which is used whenever the aspect source flag for a marker attribute is **PEXIndividual** rather than **PEXBundled**.)

Direct color components that are out of range are treated in an implementation-dependent way. If you specify a color type that is unsupported, then the implementation issues a *BadPEXColorType* error. If the color type is **PEXColorTypeIndexed** and the color index is 65535, then the implementation issues a *BadValue* error.

Valid marker bundle indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.12. Marker Glyph Table

Each entry of a marker glyph table defines a marker type.

Table Value Mask	Description	Fallback	PEX
PEXLUTVMarkerGlyphType			5.2
	marker glyph type (page 357)	**PEXMarkerTypeAsterisk**[a]	
	size of data	n/a	
	marker glyph data	n/a	
PEXLUTVMarkerGlyphAll5_2 (selects entire table entry)			5.2

a. Registered marker type **PEXMarkerTypeAsterisk** of enumerated type **PEXMarkerType**.

The superseded requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on marker glyph tables.

The implementation accesses the marker glyph table whenever a marker is rendered. Each entry of a marker glyph table consists of a marker glyph type and the data for that glyph.

PEXlib defines the following values:

marker glyph type: The value of the pipeline state’s *marker glyph type* selects the desired entry of the table.

> **PEXMarkerGlyphTypeStroke** - defines the marker glyph through a list of unconnected 2D polylines. The vertices are specified in marker normalized coordinates space which extends from -1.0 to 1.0 in both directions. The glyph is oriented at the marker position by mapping the origin of the glyph to the marker position. Marker normalized coordinates space maps to the nominal marker size. A single-coordinate polyline is rendered as a point.

If you specify an unsupported marker glyph type or if the data for a glyph of type **PEXMarkerGlyphTypeStroke** contains a polyline with no coordinates, then the implementation issues a *BadValue* error.

Valid marker glyph table indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.13. Pattern Table

Each entry of a pattern lookup table holds a pattern definition.

Table Value Mask	Description	Fallback	PEX
PEXLUTVPatternDesc			5.0
	color type	(imp-dep)	
	number of columns (*m*)	(imp-dep)	
	number of rows (*n*)	(imp-dep)	
	cell colors	(imp-dep)	
PEXLUTVPatternAll5_1 (selects entire table entry)			5.0
PEXLUTVPatternAll5_2 (selects entire table entry)			5.0

The pattern is a grid of equally sized cells. Each cell has a single color. There are m x n cells. When applying interior style **PEXInteriorStylePattern** to a fill area primitive, the pattern reference point and vectors are projected along a normal vector onto the plane of each facet of the primitive. A parallelogram is formed with its origin defined by the projected reference point. One side of the parallelogram is defined by a vector which has a direction parallel to the first projected reference vector and which has a length equal to the absolute value of the x-component of the pattern size. The adjacent side of the parallelogram is defined similarly by the second projected reference vector and the absolute value of the y-component of the pattern size. The pattern is applied across the parallelogram such that the first cell has the parallelogram origin in one corner. The first m colors in the pattern correspond to the row of cells nearest to and along the first projected reference vector. The pattern is mapped onto the facet by replicating the pattern parallelogram in directions parallel to the parallelogram's sides until the entire facet is covered.

The implementation accesses the pattern table during rendering whenever the pipeline state's *interior style* or *bf interior style* is **PEXInteriorStylePattern**, and the **PEXInteriorStylePattern** interior style is supported by the implementation. Determine the list of supported interior styles by inquiring the *InteriorStyle* enumerated type values with request **PEXGetEnumeratedTypeInfo**. When interior style **PEXInteriorStylePattern** is supported and selected, the values of the pipeline state's *interior style index* and *bf interior style index* select the desired entries of the pattern table.

If the implementation does not support the **PEXInteriorStylePattern** interior style, then it is not required to support the pattern table for use with a renderer resource. If the pattern table is not supported, **PEXCreateLookupTable**, **PEXGetTableInfo** and **PEXGetPredefinedEntries** generate a *LookupTable* error.

Direct color components that are out of range are treated in an implementation-dependent way. If you specify an unsupported color type, then the implementation issues a *BadPEXColorType* error. If the color type is **PEXColorTypeIndexed** and the color index is 65535, then the implementation issues a *BadValue* error.

Valid pattern table indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.14. Text Bundle Table

Text bundle lookup tables contain collections of text attribute values.

Table Value Mask	Description	Fallback	PEX
PEXLUTVTextBundleFontIndex			5.0
	text font index (page 407)	1	
PEXLUTVTextBundlePrecision			5.0
	text precision (page 410)	**PEXStringPrecision**	
PEXLUTVTextBundleCharExpansion			5.0
	char expansion factor (page 299)	1.0	
PEXLUTVTextBundleCharSpacing			5.0
	character spacing (page 302)	0.0	
PEXLUTVTextBundleColor			5.0
	text color (page 404)	(**PEXColorTypeIndexed**, 1)	
PEXLUTVTextBundleAll5_1 (selects entire table entry)			5.0
PEXLUTVTextBundleAll5_2 (selects entire table entry)			5.0

The implementation accesses the text bundle table during rendering whenever the aspect source flag of one of the text attributes stored in the text bundle table is **PEXBundled**. The aspect source flag for each attribute is a member of the pipeline state. The value of the pipeline state's *text bundle index* selects the desired entry of the table. The value of the bundled text attributes in that entry is then used to set the values of those text attributes in the pipeline state.

For more information about each text attribute, see the corresponding text attribute output command. (These output commands set the pipeline state which is used whenever the aspect source flag for a text attribute is **PEXIndividual** rather than **PEXBundled**.)

Direct color components that are out of range are treated in an implementation-dependent way. If you specify an unsupported color type, then the implementation issues a *PEXBadColorType* error. If the color type is **PEXColorTypeIndexed** and the color index is 65535, then the implementation issues a *BadValue* error. If you specify an unsupported text precision or if a text font index of zero, then the implementation issues a *BadValue* error.

Valid text bundle indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.15. Text Font Table

Each entry of a text font table defines a font group.

Table Value Mask	Description	Fallback	PEX
PEXLUTVTextFontGroup			5.0
	number of font identifiers (*n*)	(imp-dep)[a]	
	font identifiers	(imp-dep)[a]	
PEXLUTVTextFontAll5_1 (selects entire table entry)			5.0
PEXLUTVTextFontAll5_2 (selects entire table entry)			5.0

a. The fallback must select some font capable of representing character set ISO-8859/1 (Latin1).

A font group is a numbered list, starting at one, of resource IDs of fonts. These fonts must be in the list of fonts supported by your PEX implementation. To determine the list of fonts supported, invoke the requests **PEXListFonts** (page 712) and **PEXListFontsWithInfo** (page 713). If one of the fonts is also supported by the X server, then it does not matter whether you created the font assigned to a font group by invoking the request **PEXLoadFont** (page 714) or the X11 request **OpenFont**. The same font resource ID may appear any number of times in a font table and within a font group. Use the null resource ID **None** to "skip" font group entries or to indicate missing fonts.

The implementation accesses the text font bundle table whenever a text or annotation text primitive is rendered. The value of the pipeline state's *text font index* selects the desired entry of the table. The *char_set* value of each mono-encoding of the primitive then selects the desired member of the font group.

If a font resource referred to in a font table is closed, then the value *AlreadyFreed* is returned for the font ID in response to **PEXGetTableValues** and **PEXGetTableEntries**.

If a text font identifier doesn't name a supported X11 or PEX font resource or the null resource **None**, then the implementation issues a *BadValue* error.

Valid text font table indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.16. Texture Mapping Table

Each entry of a texture mapping table binds a set of attributes to a referenced texture map resource. The same texture map resource may be referenced by multiple texture mapping table entries.

Table Value Mask	Description	Fallback	PEX
PEXLUTVTMIDs			5.2
	number of resources (*n*)	0	
	list of texture map resources		
PEXLUTVTMRenderingControl			5.2
	rendering control	**PEXTMRCPostLight**	
PEXLUTVTMParameterizationMethod			5.2
	parameterization method	**PEXTMPMReflectionSphereVRC**	
PEXLUTVTMFPDataIndex[a]			5.2
	floating-point data index	0	
PEXLUTVTMReflectionMatrix[b]			5.2
	reflection matrix	Identity	
	reflection matrix hints	**PEXMHIdentity**	
PEXLUTVTMOrientationMatrix			5.2
	orientation matrix	Identity	
	orientation matrix hints	**PEXMHIdentity**	
PEXLUTVTMCompositionMethod			5.2
	composition method	**PEXTMCMReplace**	
PEXLUTVTMChannelSelector[c]			5.2
	channel selector	**PEXTMChannelRed**	
PEXLUTVTMEnvironmentColor[d]			5.2
	environment color	{**PEXColorTypeRGB**, (0.0, 0.0, 0.0)}	
PEXLUTVTMBackgroundColor[e]			5.2
	background color	{**PEXColorTypeRGB**, (0.0, 0.0, 0.0)}	
PEXLUTVTMSampleMethod			5.2
	minification sample method	**PEXTMSMSingleBase**	
	magnification sample method	**PEXTMSMSingleBase**	
PEXLUTVTMBoundaryConditions			5.2
	t0 boundary condition	**PEXTMBoundaryWrap**	
	t1 boundary condition	**PEXTMBoundaryWrap**	
	t2 boundary condition	**PEXTMBoundaryWrap**	
PEXLUTVTMClampMethod[f]			5.2
	boundary clamp method	**PEXTMBoundaryClampExplicitDiscontinue**	
PEXLUTVTMClampColor[g]			5.2
	boundary clamp color	{**PEXColorTypeRGB**, (0.0, 0.0, 0.0)}	
PEXLUTVTMSamplingHints			5.2
	depth sampling bias hint	0.0	
	t0 frequency hint	1.0	
	t1 frequency hint	1.0	

Table Value Mask	Description	Fallback	PEX
	t2 frequency hint	1.0	
PEXLUTVTMAll5_2 (selects entire table entry)			5.2

a. Value *fp data index* is ignored unless value *parameterization method* selects method *FloatData*.
b. Value *reflection matrix* is ignored unless value *parameterization method* selects method *ReflectionSphereVRC* or *ReflectionSphereWC*.
c. Value *channel number* is ignored unless value *composition method* selects method *BlendEnvColor*.
d. Value *environment color* is ignored unless value *composition method* selects method *BlendEnvColor*.
e. Value *background color* is ignored unless value *composition method* selects method *DecalBackground*.
f. Value *clamping method* is ignored unless value *boundary conditions* selects condition *ClampExplicitMethod*.
g. Value *clamp color* is ignored unless value *boundary conditions* selects condition *ClampExplicitMethod* and value *clamping method* selects method *Color*.

Selecting and Applying a Texture

The texture mapping table is accessed when rendering surface primitives if the pipeline state's *interior style* is **PEXInteriorStyleTexture** and the pipeline state's *active textures* field is non-empty. If the pipeline state's *facet distinguish flag* is **True**, then the implementation accesses the texture mapping table when rendering back-facing facets if the pipeline state's *bf interior style* is **PEXInteriorStyleTexture** and the pipeline state's *bf active textures* field is non-empty. The active textures lists select entries of the texture mapping table which each represent a texture map and an associated set of attributes. If the number of entries in the active textures list exceeds the maximum number of texture maps which can be applied to a single facet, then the implementation ignores the texture maps beyond the first *MaxTextureMaps* indices. Invoke request **PEXGetImpDepConstants** to inquire *MaxTextureMaps*. If the active textures list is empty, then the default table values are used to apply the single default texture map.

Not every texture map applies all of the texture mapping table fields. The descriptions of the values below indicate if, and how, the values are applied to the texture map.

Texture mapping table value *texture map ids* is used to reference the texture map resources which define the texture map. Texture map resources are discussed in "Texture Mapping" on page 617. The number of texture map resources needed to define a single texture map is determined by the texture mapping table's *parameterization method* field. All currently defined parameterization methods use a single texture map resource to define a texture map.

If the *texture map ids* fallback (an empty list) is used, then the implementation uses the default texture map; a black and white checkerboard of 8x8 squares. The checkerboard is defined by a 2D array of 16x16 texels, with each square defined by 2x2 texels. The square at $t0=t1=0.0$ is white.

rendering control: The texture mapping table's *rendering control* value controls where in the rendering pipeline the texture map is applied. The currently defined values, shown below, interact heavily with the composition method, surface interpolation method and texture mapping sample frequency. This controls whether the texture colors change the effect of lighting on a surface primitive, or if the lighting is performed on the textured surface. When a texture is applied post-light, it can reduce or obliterate any specular highlight; but if texture is applied pre-light, then the specular highlight is retained.

PEXTMRenderingControlPreLight - applies the texture map to the primitive before the reflectance calculations.

PEXTMRenderingControlPostLight - applies the texture map to the primitive after the reflectance calculations.

See "Composition Controls" on page 465.

Parameterization Controls

The parameterization attributes control how the texture map coordinates are derived. There are texture map coordinates associated with each point on the surface for which a texture map value is sampled. They are used to select the point within the texture map which is sampled. There is one texture map coordinate for each dimension of the texture map. For example, three texture map coordinates, represented as ($t0$, $t1$, $t2$), are used to sample a 3D texture map. The texture coordinates are normalized, that is, they range from 0.0 to 1.0 across the texture map.

parameterization method: The *parameterization method* field of the texture mapping table defines how texture map coordinates are derived: whether explicitly defined by the primitive's data, calculated through a projection mapping, or determined through some other means. PEXlib defines the following parameterization methods:

The **PEXTMParamReflectionSphereVRC** and **PEXTMParamReflectionSphereWC** projection mapping parameterization methods project a reflection vector (the 3D source texture map coordinates) onto a sphere (the projection object) to determine the 3D texture map coordinates ($t0$,$t1$,$t2$=0.0).

The spheres have fixed orientations. The position of the reflection vectors relative to these orientations may be modified through the multiplication of the reflection vectors by the reflection matrix from the texture mapping LUT. The reflection vectors are always multiplied by the reflection matrix before the reflection is performed. For best results the reflection transformation should utilize only rotations. The reflection matrix hints provide an opportunity for optimizing this transformation. See "PEXOCCSetGlobalTransform - Set Global Transformation 3D (5.0)" on page 316 for an explanation of matrix hints.

PEXTMParamVertexMCCoord[1] - Sets the texture map coordinate values equal to the MC space values of the vertex coordinates: $t0 = x$; $t1 = y$; $t2 = z$.

PEXTMParamVertexWCCoord[1]- Sets the texture map coordinate values equal to the WC space values of the vertex coordinates: $t0 = x'$; $t1 = y'$; $t2 = z'$.

PEXTMParamVertexMCNormal[1] - Sets the texture map coordinates equal to the MC space value of the vertex normal. If a normal is not explicitly supplied with the vertex, then the calculated geometric normal is used instead. $t0 = nx$; $t1 = ny$; $t2 = nz$.

PEXTMParamVertexWCNormal[1] - Sets the texture map coordinates equal to the WC space value of the vertex normal. If a normal is not explicitly supplied with the vertex, then the calculated geometric normal is used instead. $t0 = nx'$; $t1 = ny'$; $t2 = nz'$.

PEXTMParamNURBUV - Sets the texture map coordinates equal to the *u* and *v* parameters of the non-uniform b-spline surface: $t0 = u$; $t1 = v$; $t2 = 0.0$.

If the primitive is not a non-uniform b-spline surface, then the texture map coordinates are set to: $t0 = t1 = t2 = 0.0$.

PEXTMParamFloatData - The texture map coordinates are members of the vertex's floating-point data list. The *fp data index* value of the texture mapping table is the index into the floating-point data list of the value which is assigned to $t0$. The subsequent values in the floating-point data list are assigned to $t1$ and $t2$:

$t0$ = vertex_fp_data[*fp data index*]
$t1$ = vertex_fp_data[*fp data index* + 1] (if 2D map)
$t2$ = vertex_fp_data[*fp data index* + 2] (if 3D map)

1. In the case of rendering primitives which do include explicit vertices, but may have computed vertices, the behavior is implementation-dependent. The implementation may use the computed vertices or may evaluate the coordinate or normal at each pixel.

If the primitive is a non-uniform b-spline surface, then the values are obtained from the spline's data splines:

$t0$ = first value of the data spline selected by the
fp data index
$t1$ = first value of the data spline selected by
fp data index + 1
$t2$ = first value of the data spline selected by
fp data index + 2

If the number of floating-point data values is insufficient for the dimension of the texture map, then the texture coordinate values used are implementation-dependent.

PEXTMParamReflectionSphereVRC - The infinite sphere has (0,0,0) as its origin and the $+y$ axis as its axis of revolution. The texture seams lie on the positive and negative x axes. The source texture map coordinates are a reflection vector ($s0$, $s1$, $s2$), computed to determine a point on the interior of the sphere.

The reflection vector is calculated as follows: $s = i - 2n(n \cdot i)$

where s is the reflection vector, n is the vertex normal (explicit or calculated), and i is the unit vector from the eyepoint (computed as described in "View Table" on page 472) to the vertex. This calculation is performed in VRC space, after the vertex and its normal have been transformed to VRC space. After this, the texture coordinates are derived as follows:

$$t_0 = \frac{s_0}{m} + \frac{1}{2}$$

$$t_1 = \frac{s_1}{m} + \frac{1}{2}$$

where:

$$m = 2 \times \sqrt{s_0^2 + s_1^2 + (s_2 + 1)^2}$$

PEXTMParamReflectionSphereWC - The source texture map coordinates, a reflection vector ($s0$,$s1$,$s2$), are normalized and conceptually projected onto the infinite sphere (with (0,0,0) as its origin) projection object. The axis of revolution of the sphere is the $+y$ axis (in WC space). The texture seam sweeps from the $+x$ axis (in WC space) in a counterclockwise direction: [0..2PI].

The reflection vector is calculated as follows:

$$s = -(i - 2n(n \cdot i))$$

where s is the reflection vector, n is the vertex normal (explicit or calculated), and i is the unit vector from the eyepoint (computed as described in "View Table" on page 472) to the vertex. This calculation is performed in WC space, after the vertex and its normal have been transformed to WC space. This parameterization method is primarily intended for use with a perspective, as opposed to orthogonal, view projection. After the reflection vector is calculated, the texture coordinates are derived as follows.

$$t_0 = 0.75 \ : s_0 = 0.0, s_2 > 0.0$$
$$t_0 = 0.25 \ : s_0 = 0.0, s_2 \le 0.0$$
$$t_0 = \frac{-\mathrm{atan}\left(\frac{s_2}{s_0}\right)}{2\pi} + 0.5 \ : s_0 < 0.0$$
$$t_0 = \frac{-\mathrm{atan}\left(\frac{s_2}{s_0}\right)}{2\pi} + 1.0 \ : s_0 > 0.0, s_2 > 0.0$$
$$t_0 = \frac{-\mathrm{atan}\left(\frac{s_2}{s_0}\right)}{2\pi} + 0.0 \ : s_0 > 0.0, s_2 \le 0.0$$

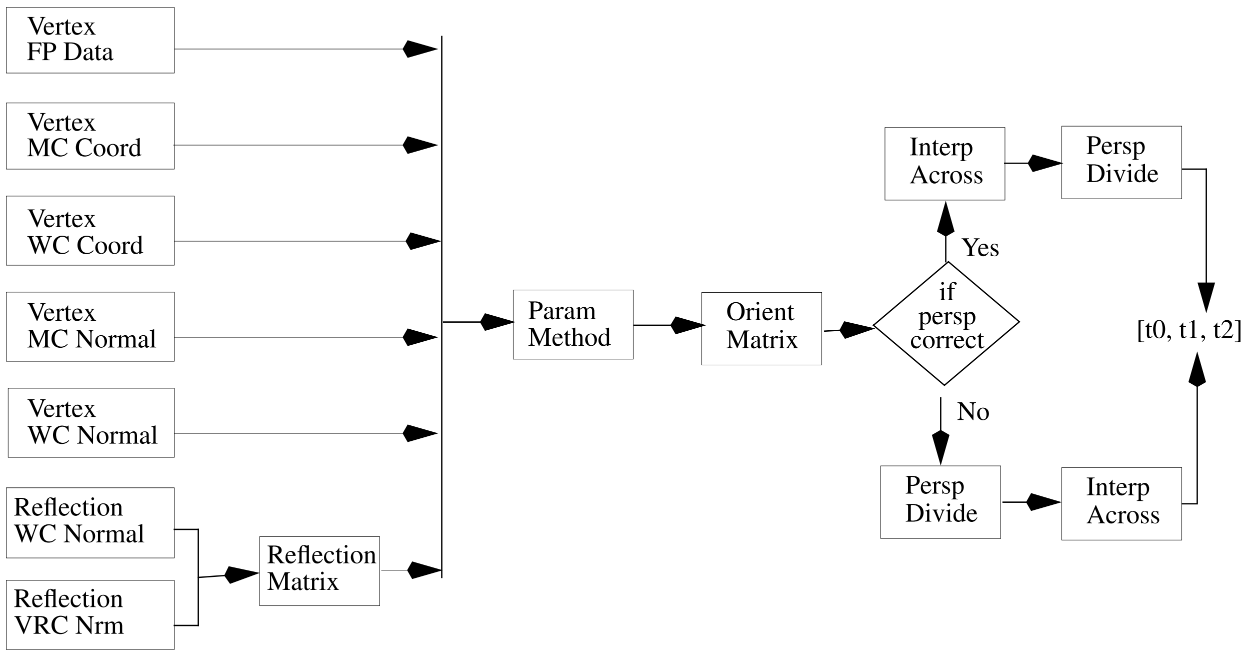
$$t_1 = \frac{\mathrm{asin}\,(s_1)}{\pi} + 0.5$$

where: $(\frac{-\pi}{2} \le \mathrm{atan}\,(r) \le \frac{\pi}{2})$ and $(\frac{-\pi}{2} \le \mathrm{asin}\,(r) \le \frac{\pi}{2})$

Orientation Transformation, Interpolation, and Perspective Correction

Once the texture map coordinates are determined, independent of which parameterization method was used, they are transformed by the *orientation matrix* field of the texture mapping table. The orientation matrix hints provide an opportunity to optimize the transformation. See "PEXOCCSetGlobalTransform - Set Global Transformation 3D (5.0)" on page 316 for an explanation of matrix hints. The orientation transformation occurs before any interpolation of the texture map coordinates across the facet that may be required before the coordinates are used to sample the texture map. If the projection is perspective, then the interpolated texture map coordinates are also adjusted according to the pipeline state's *tm perspective correction* before the sampling is done."Texture Parameterization" on page 465 provides a graphical description of parameterization.

Table 13: Texture Parameterization



Composition Controls

The currently defined composition attributes control how the texture map color interacts with the primitive's color to determine the primitive's color after the texture map is applied.

The texture mapping table *composition method* value specifies how the texture map is blended with the primitive's existing data. The currently defined composition methods assume:

- The texture is a color domain texture, i.e. it modifies intrinsic color (including alpha) of rendered primitives.
- The rendering color model is RGB. The current methods have undefined behavior for rendering color models other than RGB.

For each composition method, equations are given. The equations given are for the input into each texture map *i* and show the C_{out} and A_{out} for each texture map *i*. The output of each texture map *i* becomes the input of the next texture map *i*+1. The final output of the texture mapping stage is the output of texture map *n*, where there are *n* texture maps active for the stage. In some *PreLight* rendering control cases there are additional values used as input into each texture map *i* (e.g., C_W) and used as output from the texture mapping stage (e.g., C_S). Table 17, "Composition Method BlendEnvColor," on page 468 defines the symbols used to define the composition methods.

Table 14: Symbols Used in Defining the Composition Methods

Symbol	Description
in	Primitive color components input into texture *i* of a texture mapping stage.
out	Primitive color components output from texture *i* of a texture mapping stage.
R	The red component.
B	The blue component.
G	The green component.

Table 14: Symbols Used in Defining the Composition Methods

Symbol	Description
C	*R, G,* and *B* considered together.
A	The alpha component.
L	The luminance component.
C_W	A portion of primitive color input into the *PreLight* texture mapping stage. C_W is derived in the preceding lighting stage by applying the ambient and diffuse terms of the reflectance equation assuming the primitive's color is white. The same value of C_W is input into each texture i of the texture mapping stage; it is not modified by texture mapping.
C_D	Another portion of primitive color input into the *PreLight* texture mapping stage. C_D is the ambient and diffuse components of color derived in the preceding lighting stage by applying the reflectance equation using the primitive's color (as opposed to assuming the primitive is white).
C_S	A third portion of primitive color input into the *PreLight* texture mapping stage. C_S is the specular component of color derived in the preceding lighting stage by applying the ambient and diffuse terms of the reflectance equation using the primitive's color (as opposed to assuming the primitive is white). C_S is not affected by the texture mapping stage. It is combined with the C_D output from the texture mapping stage to form the RGB components of the primitive color output from the texture mapping stage.
C_T	*R, G,* and *B* of the texture map color considered together. If the texels have a luminance component instead of *RGB* components, $R_T = G_T = B_T = L$ 1
A_T	The alpha component of the texture map color.
C_E	When the composition method is *BlendEnvColor*, C_E is taken from the environment color field of the texture mapping lookup table.
C_B	When the composition method is *DecalBackground*, C_B is taken from the background color field of the texture mapping lookup table.
Γ	When the composition method is *Decal*, Γ is taken from the texture map color. For texels defined with *RGB* or *RGBA* components: if the texture mapping lookup table's *channel number* is *TMChannelRed*, $\Gamma = R_T$; if *channel number* is *TMChannelGreen*, $\Gamma = G_T$; if *channel number* is *TMChannelBlue*, $\Gamma = B_T$, and if there is an alpha component and *channel number* is *TMChannelAlpha*, $\Gamma = A_T$. For texels defined with a luminance component: $\Gamma = L$ The channel number constants are defined in "Channel Selection Constant Values" on page 469.

PEXlib defines the following composition methods:

PEXTMCompReplace - The texture map replaces the primitive's existing data. The texture map data unconditionally replaces the primitive's color. See Table 17, "Composition Method BlendEnvColor," on page 468 for more details.

Table 15: Composition Method *Replace*

<table>
<tr><th>Texel Components</th><th>Rendering Control</th><th>Interpolation Method</th><th>Sample Frequency</th><th>C_{out}</th><th>A_{out}</th></tr>
<tr><td rowspan="4">*RGB*, *L*</td><td rowspan="3">PreLight</td><td>None, Color</td><td>pixel</td><td>$C_W \bullet C_T$ and C_s</td><td>1.0</td></tr>
<tr><td>None, Color</td><td>InterpDep</td><td rowspan="3">C_T</td><td rowspan="3">1.0</td></tr>
<tr><td>DotProduct, Normal</td><td>(any)</td></tr>
<tr><td>PostLight</td><td>(any)</td><td>(any)</td></tr>
<tr><td rowspan="4">*RGBA*, *LA*</td><td rowspan="3">PreLight</td><td>None, Color</td><td>pixel</td><td>$C_W \bullet C_T$ and C_s</td><td>A_T</td></tr>
<tr><td>None, Color</td><td>InterpDep</td><td rowspan="3">C_T</td><td rowspan="3">A_T</td></tr>
<tr><td>DotProduct, Normal</td><td>(any)</td></tr>
<tr><td>PostLight</td><td>(any)</td><td>(any)</td></tr>
</table>

PEXTMCompModulate - The texture map color and primitive color are blended. See Table 17, “Composition Method BlendEnvColor,” on page 468 for more details.

Table 16: Composition Method *Modulate*

<table>
<tr><th>Texel Components</th><th>Rendering Control</th><th>Interpolation Method</th><th>Sample Frequency</th><th>C_{out}</th><th>A_{out}</th></tr>
<tr><td rowspan="4">*RGB*, *L*</td><td rowspan="3">PreLight</td><td>None, Color</td><td>pixel</td><td>$C_D \bullet C_T$ and C_s</td><td>A_{in}</td></tr>
<tr><td>None, Color</td><td>InterpDep</td><td rowspan="3">$C_{in} \bullet C_T$</td><td rowspan="3">A_{in}</td></tr>
<tr><td>DotProduct, Normal</td><td>(any)</td></tr>
<tr><td>PostLight</td><td>(any)</td><td>(any)</td></tr>
<tr><td rowspan="4">*RGBA*, *LA*</td><td rowspan="3">PreLight</td><td>None, Color</td><td>pixel</td><td>$C_D \bullet C_T$ and C_s</td><td>$A_{in} \bullet A_T$</td></tr>
<tr><td>None, Color</td><td>InterpDep</td><td rowspan="3">$C_{in} \bullet C_T$</td><td rowspan="3">$A_{in} \bullet A_T$</td></tr>
<tr><td>DotProduct, Normal</td><td>(any)</td></tr>
<tr><td>PostLight</td><td>(any)</td><td>(any)</td></tr>
</table>

PEXTMCompBlendEnvColor - The texture map blends with the primitive’s existing environment color. The primitive color and a constant environment color are blended by the texture map color. The primitive alpha is blended with the texture map alpha, if the texture map has alpha. See Table 17, “Composition Method

BlendEnvColor," on page 468 for more details.

Table 17: Composition Method *BlendEnvColor*

Texel Components	Rendering Control	Interpolation Method	Sample Frequency	C_{out}	A_{out}
RGB, *L*	PreLight	None, Color	pixel	$C_D(1-\Gamma)+C_E\Gamma$ and C_s	A_{in}
		None, Color	InterpDep	$C_{in}(1-\Gamma)+C_E\Gamma$	A_{in}
		DotProduct, Normal	(any)		
	PostLight	(any)	(any)		
RGBA, *LA*	PreLight	None, Color	pixel	$C_D(1-\Gamma)+C_E\Gamma$ and C_s	$A_{in} \bullet A_T$
		None, Color	InterpDep	$C_{in}(1-\Gamma)+C_E\Gamma$	$A_{in} \bullet A_T$
		DotProduct, Normal	(any)		
	PostLight	(any)	(any)		

PEXTMCompDecal - The texture map color and primitive color are blended by the texture map alpha. If the texture map doesn't contain alpha, then the texture map color replaces the primitive color. In either case, the primitive's alpha is unchanged. See Table 17, "Composition Method BlendEnvColor," on page 468 for more details.

Table 18: Composition Method *Decal*

Texel Components	Rendering Control	Interpolation Method	Sample Frequency	C_{out}	A_{out}
RGB, *L*	PreLight	None, Color	pixel	$C_W \bullet C_T$ and C_s	A_{in}
		None, Color	InterpDep	C_T	A_{in}
		DotProduct, Normal	(any)		
	PostLight	(any)	(any)		
RGBA, *LA*	PreLight	None, Color	pixel	$C_D(1-A_T)+C_W \bullet C_T \bullet A_T$ and C_s	A_{in}
		None, Color	InterpDep	$C_{in}(1-A_T)+C_T A_T$	A_{in}
		DotProduct, Normal	(any)		
	PostLight	(any)	(any)		

PEXTMCompDecalBackground - The texture map color and background color are blended according to the texture map alpha, and this blend replaces the primitive color. If the texture map doesn't contain alpha, then the texture map color replaces the primitive color. In either case, the primitive's alpha is unchanged. See Table 17, “Composition Method BlendEnvColor,” on page 468 for more details.

Table 19: Composition Method *DecalBackground*

Texel Components	Rendering Control	Interpolation Method	Sample Frequency	C_{out}	A_{out}
RGB, *L*	PreLight	None, Color	pixel	$C_W \bullet C_T$ and C_s	A_{in}
		None, Color	InterpDep	C_T	A_{in}
		DotProduct, Normal	(any)		
	PostLight	(any)	(any)		
RGBA, *LA*	PreLight	None, Color	pixel	$C_B(1 - A_T) + C_W \bullet C_T \bullet A_T$ and C_s	A_{in}
		None, Color	InterpDep	$C_B(1 - A_T) + C_T A_T$	A_{in}
		DotProduct, Normal	(any)		
	PostLight	(any)	(any)		

Channel Selection Constant Values

PEXlib defines the following channel selectors:

PEXTMChannelRed - selects the red channel, channel 0.

PEXTMChannelGreen - selects the green channel, channel 1.

PEXTMChannelBlue - selects the blue channel, channel 2.

PEXTMChannelAlpha - selects the alpha channel, channel 3.

Sampling Controls

The sampling attributes are used to determine how a texture map color is derived when the texture map is sampled.

Minification and Magnification Sample Methods

The *sample min and mag* field of the texture mapping table defines the minification and magnification sample methods. These methods specify how texels are sampled from the texture map and mapped to primitive pixels when the area covered by a texel and pixel differ.

In the case of a pixel overlapping more than one texture map texel, the texture minification method indicates how the sample value is derived for that pixel.

In the case of a pixel overlapping less than one texture map texel, the texture magnification method indicates how the sample value is derived for that pixel.

PEXlib defines the following sample methods:

PEXTMSampleSingleBase - The closest single texel selected by the texture coordinate(s) is sampled from the base texture map level.

PEXTMSampleWeightedBase - The 2^n (where n is the dimension of the texture map) closest texels selected by the texture coordinate(s) are sampled from the base texture map level. Their weighted average is used.

PEXTMSampleSingleInMipmap - The closest single texel selected by the texture coordinate(s) is sampled from the closest texture map level to the sample depth.

PEXTMSampleWeightedInMipmap - The 2^n (where n is the dimension of the texture map) closest texels selected by the texture coordinate(s) are sampled from the closest texture map level to the sample depth. Their weighted average is used.

PEXTMSampleSingleBetweenMipmaps - The closest texel selected by the texture coordinate(s) is sampled from the two closest texture map levels to the sample depth. The texel found at the exact sample depth by interpolation between these two sampled texels is used. If the sample depth is beyond the base or pinnacle texture map levels, that level is used and this method behaves the same as **PEXTMSampleSingleInMipmap**.

PEXTMSampleWeightedBetweenMipmaps - The 2^n (where n is the dimension of the texture map) closest texels selected by the texture coordinate(s) are sampled from the two closest texture map levels to the sample depth. A weighted average is taken of these 2^n texels on each of the texture map levels. The texel found at the exact sample depth by interpolation between these two calculated texels is used. If the sample depth is beyond the base or pinnacle texture map levels, that level is used and this method behaves the same as **PEXTMSampleWeightedInMipmap**.

Boundary Conditions

The *boundary conditions* field of the texture mapping table specifies the texturing to be applied to the primitive when the texture coordinates ($t0$, $t1$, $t2$) select a point beyond a boundary of the texture map. That is, the calculated texture coordinates are outside the range [0.0,1.0]. The boundary condition in each direction is independently controlled.

PEXlib defines the following boundary conditions:

PEXTMBoundaryClampExplicit - applies the method specified by the *clamping method* field of the texture mapping table.

PEXTMBoundaryClampBorder - applies the closest boundary texel.

PEXTMBoundaryWrap - texel sampling wraps back to the opposite texture map border, creating a “tiling” or “repeating” effect of the texture map across the primitive.

PEXTMBoundaryMirror - texel sampling is reversed across the texture map, creating a “mirrored alternating” pattern of the texture map across the primitive.

Clamping Method

The *clamping method* field of the texture mapping table specifies whether to discontinue texturing or to use an explicit color when *boundary condition ClampExplicitMethod* is selected. Although these are seemingly really additional boundary conditions, they are controlled through a separate field of the texture mapping table by design. With this level of direction, it is always clear what the behavior should be when a location is beyond the texture map in more than one direction, even if the boundary conditions for those directions differ.

PEXlib defines the following boundary clamp methods:

PEXTMBoundaryClampExplicitDiscontinue - discontinues texturing. That is, it maintains the primitive’s appearance before applying the current texture map.

PEXTMBoundaryClampExplicitColor - applies the color specified by the *clamp color* field of the texture mapping table.

Sampling Hints

Use the texture mapping table *sampling hints* value to adjust the selected sampling location. The depth bias hint is used to adjust the sampling depth. A bias of -1.0 moves the sampling depth one level toward the texture map's base level, effectively making the texturing more detailed, or jaggy, than it would otherwise have been. A bias of 2.0, for example, moves the sampling depth away from the texture map's base level by two levels, effectively blurring the texturing. The following are the suggested formulae for implementations; PEX implementations may use other formulae which produce a similar effect.

$$A = max\,(t_0 freq \times \frac{dt_0}{dx}, t_1 freq \times \frac{dt_1}{dx})$$

$$B = max\,(t_0 freq \times \frac{dt_0}{dy}, t_1 freq \times \frac{dt_1}{dy})$$

$$depth = bias + 0.5 \times \log 2\,(A^2 + B^2)$$

The extension of the suggested formulae to three-dimensional texture maps is left unspecified.

The suggested depth bias formulae utilize the frequency hints. There is a frequency hint corresponding to each direction of the texture map. A frequency hint is set to something less than 1.0, but greater than 0.0, if blurring in that direction would be unacceptably high due to low spatial frequency. A value of 1.0 for the corresponding texture map coordinate's frequency hint indicates the spatial frequency isn't high. If the spatial frequency of the texture map data is unknown, then it is best to set the values to 1.0.

Inquire *TMSamplingHintsUsed* with request **PEXGetImpDepConstants** to determine if the sampling hints are used.

Restrictions and Errors

The superseded requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on texture mapping tables.

Valid texture mapping table indices are in the range [1, 65535]. The fallback entry is entry 1.

10.1.17. View Table

Each entry of a view table holds a view definition.

Table Value Mask	Description	Fallback	PEX
PEXLUTVViewClipFlags			5.0
	clip flags	**PEXClipAll**	
PEXLUTVViewClipLimits			5.0
	clip limits	(0,0,0)-(1,1,1)	
PEXLUTVViewOrientationMatrix			5.0
	view orientation matrix	Identity	5.0
	view orientation matrix hints[a]	Identity	5.2
PEXLUTVViewMappingMatrix			5.0
	view mapping matrix	Identity	5.0
	view mapping matrix hints[a]	Identity	5.2
PEXLUTVViewEyepointHint[b]			5.2
	eyepoint hint	(**PEXViewEyepointImplicit** ,n/a)	
PEXLUTVViewAll5_1 (selects entire table entry except 5.2 items)			5.0
PEXLUTVViewAll5_2 (selects entire table entry)			5.2

a. The requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on the hints portion of the matrix field.
b. The requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on the eyepoint value of a view table entry.

clip_flags contains three bits that indicate whether primitives should be clipped against the sides, back, and front planes of the volume specified by *clip_limits*. The *clip_limits* specify the minimum and maximum of a rectangular volume in NPC's. Together these define the view clipping volume. The actual clipping volume for each view is a rectangular parallelpiped that is computed by taking the intersection of the view clipping volume and the NPC subvolume.

The *orientation_matrix* defines the mapping from WC's to VRC's. The *mapping_matrix* defines the mapping from VRC's to NPC's. Together the *orientation_matrix* and *mapping_matrix* define the view transformation.

eyepoint hint is a hint describing the definition (in WC's) of the eyepoint used with the view. *EyepointUsed,* inquired with the function **PEXGetImpDepConstants**, indicates whether the implementation uses the eyepoint hint specified.

PEXlib defines the following matrix hints:

> **PEXMHNone**, **PEXMHIdentity**, **PEXMHTranslation**, **PEXMHRotation**, **PEXMHReflection**, **PEXMHUniformScale**, **PEXMHNonUniformScale**, **PEXMHShear**, **PEXMHPerspProj**, **PEXMHNonPerspProj**, **PEXMH3DTo2DParProj**, **PEXMH3DTo2DPerspProj**, **PEXMHZero**, **PEXMHSingularUnknown**, **PEXMHNonSingularUnknown**

See "PEXOCCSetGlobalTransform - Set Global Transformation 3D (5.0)" on page 316 for an explanation of matrix hints.

PEXlib defines the following eyepoint hints:

> **PEXViewEyepointImplicit** - The eyepoint is defined implicitly by the view. No additional data is defined with this method. The eyepoint is obtained by transforming a point at infinite positive *z* (NPC's), (0, 0, 1, 0), by the inverse of the view transformation. If the view matrix specifies perspective projection, a point is obtained.

If parallel projection is in use, then a direction vector is obtained. Singular view matrices lead to implementation-dependent results.

PEXViewEyepointDirection - The eyepoint is located at infinity along a particular direction. A vector (in WC's) anchored at the origin and pointing toward the eyepoint defines the eyepoint. The *Directional* method should be used when the projection is parallel. It can also be used with faster but less accurate results when the projection is perspective.

PEXViewEyepointPositional - The eyepoint is located at a particular position (in WC's). This method should be used only when the projection is perspective.

The implementation accesses the view table during rendering. The value of the pipeline state's *view index* selects the desired entry of the table.

If you specify an unsupported eyepoint specification method, if an undefined bit is set in the *clip flags* field, or if a directional eyepoint hint has a direction vector of length zero, then the implementation issues a *BadValue* error.

Valid view table indices are in the range [0, 65534]. The fallback entry is entry 0.

PEXChangeTableValues - Change Table Values (5.2)

Synopsis

void **PEXChangeTableValues**(Display **display*, `PEXLookupTable` *table*, int *table_type*, unsigned int *index*, unsigned long *mask*, `PEXPointer` *entry*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
table	The resource identifier of the lookup table [§2.3.8, page 28].
table_type	The type of lookup table (see "Table Types, Availability, and Definitions" on page 439).
index	The index of the table entry to be set.
mask	The mask indicating which table entry values are to be set.
entry	A pointer to a table values structure [§2.18.1, page 46].

Returns

None.

Description

Invoke **PEXChangeTableValues** to set the specified values in the lookup table entry *index* in the specified lookup table.

The *entry* must point to a structure that is appropriate for the type of lookup table. See "Table Types, Availability, and Definitions" on page 439 to determine the PEXlib data structure appropriate for each type of lookup table. The argument must also be set using bits appropriate to the table type.

Errors

BadAlloc	The implementation failed to allocate the resource. You may have tried to set more table entries than the implementation allows.
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	A table entry field contains an invalid value, or index 0 is invalid for the specified table type.

See Also

PEXCreateLookupTable, PEXGetDefinedIndices, PEXGetTableValues,

PEXCopyLookupTable - Copy Lookup Table (5.0)

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 [§2.3.8, page 28].
dst_table	The resource identifier of the destination lookup table [§2.3.8, page 28].

Returns

None.

Description

Invoke **PEXCopyLookupTable** to copy entries from the source lookup table (*src_table*) to the destination lookup table (*dst_table*). This function deletes all entries in the destination lookup table first.

Both tables must already exist as valid lookup table resources and both must be the same table type. You also must have created both lookup tables for the same class of drawables.

Errors

BadMatch	The specified lookup tables were not created with compatible drawables.
BadPEXLookupTable	A specified lookup table resource identifier is invalid.

See Also

PEXCreateLookupTable

PEXCreateLookupTable - Create Lookup Table (5.0)

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 "Table Types, Availability, and Definitions" on page 439).

Returns

The resource identifier of the newly-created lookup table [§2.3.8, page 28].

Description

Invoke **PEXCreateLookupTable** to create a lookup table resource of the specified type.

This function returns the resource identifier of the new lookup table. You can then specify this resource identifier to refer to the created lookup table. See "Table Types, Availability, and Definitions" on page 439 for the types of tables defined by PEXlib for creation with this function.

You can only use the newly-created lookup table with drawables having the same depth and root as the specified drawable.

Errors

BadAlloc	The implementation 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 (5.0)

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 [§2.3.8, page 28].
start	The initial table entry to be deleted.
count	The number of entries to be deleted.

Returns

None.

Description

Invoke **PEXDeleteTableEntries** to delete lookup table entries from the specified lookup table.

This function starts deleting entries at the specified entry and deletes entries with index values between *start* and (*start* + *count* -1), inclusive.

The PEX implementation ignores attempts to delete undefined table entries.

Errors

BadPEXLookupTable	The specified lookup table resource identifier is invalid.
BadValue	The sum of start and count is too large, or index 0 is invalid for the specified table type.

See Also

PEXGetDefinedIndices, **PEXGetTableInfo**

PEXFreeLookupTable - Free Lookup Table (5.0)

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 [§2.3.8, page 28].

Returns

None.

Description

Invoke **PEXFreeLookupTable** to delete the association between the lookup table resource identifier and the lookup table.

The implementation frees the lookup table when no other resource references the lookup table.

Errors

BadPEXLookupTable	The specified lookup table resource identifier is invalid.

See Also

PEXCreateLookupTable

PEXFreeTableValues - Free Storage From Get Table Values (5.2)

Synopsis

void **PEXFreeTableValues**(int *table_type*, `PEXPointer` *entries*)

Arguments

table_type	The type of table entries in the array.
entries	A pointer to a table values structure [§2.18.1, page 46].

Returns

None.

Description

Invoke **PEXFreeTableValues** to deallocate the memory returned by **PEXGetTableValues.**

Errors

None.

See Also

PEXGetDefinedIndices - Get Lookup Table Defined Indices (5.0)

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 [§2.3.8, page 28].
count_return	Returns the number of returned indices.
indices_return	Returns an array of indices to defined table entries [§2.3.21, page 29].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetDefinedIndices** to obtain the defined indices for the specified lookup table.

This function returns the index of each defined table in a list.

PEXlib allocates memory for the returned indices. Invoke **XFree** to deallocate the memory.

Errors

BadPEXLookupTable	The specified lookup table resource identifier is invalid.

See Also

PEXCreateLookupTable, **PEXGetTableInfo**

PEXGetTableInfo - Get Lookup Table Information (5.0)

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 "Table Types, Availability, and Definitions" on page 439).
info_return	Returns the lookup table information [§2.16.23, page 43].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetTableInfo** to obtain information about the specified type of lookup table. The type of lookup table must be one of the types described in "Table Types, Availability, and Definitions" on page 439.

This function assumes that you will use the lookup table on drawables with the same root and depth as the 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).

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

PEXGetTableValues - Get Table Values (5.2)

Synopsis

`PEXPointer` **PEXGetTableValues**(Display **display*, `PEXLookupTable` *table*, int *value_type*, unsigned int *index*, unsigned long *mask*, 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 [§2.3.8, page 28].
value_type	The type of values to return (**PEXSetValue** or **PEXRealizedValue**).
index	The index of the table entry from which to fetch the values.
mask	The mask indicating which table entry values are to be returned.
table_type_return	Returns the type of table.

Returns

Zero if unsuccessful; otherwise a pointer to a data structure mapping the returned table entry values, depending on table type [§2.18.1, page 46]. See "Table Types, Availability, and Definitions" on page 439 for a description.

Description

Invoke **PEXGetTableValues** to obtain the specified values from the lookup table entry *index* in the specified lookup table.

If the implementation does not define the specified table entry you request, then this function returns the values for the default entry for a table of the requested type structure types.

PEXlib defines the following *value types*:

PEXSetValue - returns the values originally set in the table entry.

PEXRealizedValue - returns the values 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 entry. Invoke **PEXFreeTableValues** to deallocate the memory.

Errors

BadPEXLookupTable	The specified lookup table resource identifier is invalid.
BadValue	Index 0 is invalid for the specified table type.

See Also

PEXChangeTableValues, PEXCreateLookupTable, PEXGetDefinedIndices

10.1.18. Lookup Table Requests for PEXlib 5.0 and 5.1 Compatibility

The compatibility functions include:

PEXFreeTableEntries - superseded by **PEXFreeTableValues** (page 479).

PEXGetPredefinedEntries -This function is not superseded. PEX 5.2 clients should not depend on predefined entries.

PEXGetTableEntries - superseded by **PEXGetTableValues** (page 482).

PEXGetTableEntry - superseded by **PEXGetTableValues** (page 482).

PEXSetTableEntries - superseded by **PEXChangeTableValues** (page 474).

PEXFreeTableEntries - Free Storage Associated with Tables Entries Return Parameters (5.0)

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 [§2.18.1, page 46].

Returns

None.

Description

Invoke **PEXFreeTableEntries** to deallocate the memory returned by the compatibility functions: **PEXGetPredefinedEntries, PEXGetTableEntries** and **PEXGetTableEntry.**

Errors

None.

See Also

PEXGetPredefinedEntries - Get Lookup Table Predefined Indices (5.0)

Note: It is recommended that the following compatibility functions not be used for PEXlib applications because of the wide variation in the predefined values found in various implementations.

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 [§2.18.1, page 46].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetPredefinedEntries** to obtain 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 you create a lookup table. This function assumes that you will use the lookup table on drawables with the same root and depth as the specified drawable. At most, this function returns the specified number of entries, starting with the specified entry index. You cannot request more entries than predefined entries. Invoke **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. Invoke **PEXFreeTableEntries** to deallocate the memory.

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, PEXGetDefinedIndices, PEXGetTableEntries, PEXGetTableEntry, PEXGetTableInfo, PEXSetTableEntries

PEXGetTableEntries - Get Lookup Table Entries (5.0)

Note: This function has been superseded by the **PEXGetTableValues** function (page 482), but it is maintained here for backwards compatibility with PEXlib 5.1.

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 [§2.3.8, page 28].
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 [§2.18.1, page 46].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetTableEntries** to obtain the values associated with a range of contiguous table indices, starting at the specified entry index.

The type of the table specified by *table* must be one of the following:

PEXLUTColorApprox
PEXLUTColor
PEXLUTDepthCue
PEXLUTEdgeBundle
PEXLUTInteriorBundle
PEXLUTLight
PEXLUTLineBundle
PEXLUTMarkerBundle
PEXLUTPattern
PEXLUTTextBundle
PEXLUTTextFont
PEXLUTView

If a table entry in the range you request is not defined, then this function returns the default entry for a table of the requested type for the undefined entry. The type of entries in the returned list depend on the returned table type. See **PEXSetTableEntries** for the structure types.

If you request a value type of **PEXSetValue**, then this function returns the values originally set in the table entry. If you request a value type of **PEXRealizedValue**, then this function returns the values actually used during rendering (i.e., the values used if the specified entry value cannot be exactly matched).

PEXlib allocates the memory for the returned entries. Invoke **PEXFreeTableEntries** to deallocate the memory.

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, PEXGetDefinedIndices, PEXGetPredefinedEntries, PEXSetTableEntries, PEXGetTableEntry, PEXGetTableInfo

PEXGetTableEntry - Get Lookup Table Entry (5.0)

Note: This function has been superseded by the **PEXGetTableValues** function (page 482), but it is maintained here for backwards compatibility with PEXlib 5.1.

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 [§2.3.8, page 28].
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 [§2.18.1, page 46]; a null pointer if unsuccessful.

Description

Invoke **PEXGetTableEntry** to obtain a single entry from the specified lookup table.

The type of the table specified by *table* must be one of the following:

PEXLUTColorApprox
PEXLUTColor
PEXLUTDepthCue
PEXLUTEdgeBundle
PEXLUTInteriorBundle
PEXLUTLight
PEXLUTLineBundle
PEXLUTMarkerBundle
PEXLUTPattern
PEXLUTTextBundle
PEXLUTTextFont
PEXLUTView

If the lookup table entry you specify is defined, then this function returns the defined entry. If the lookup table entry is not defined, then this function returns the default entry for that type of table. The type of structure returned depends on the specified type. See **PEXSetTableEntries** for the structure types.

If you request a value type of **PEXSetValue**, then this function returns the values originally set in the table entry. If you request a value of **PEXRealizedValue**, then this function returns the values 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. Invoke **PEXFreeTableEntries** to deallocate the memory.

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, PEXGetDefinedIndices, PEXGetPredefinedEntries, PEXSetTableEntries, PEXGetTableEntry, PEXGetTableInfo

PEXSetTableEntries - Set Lookup Table Entries (5.0)

Note: This function has been superseded by the **PEXChangeTableValues** (page 474), but it is maintained here for backwards compatibility with PEXlib 5.1.

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 [§2.3.8, page 28].
start	The index of the first table entry to be set.
count	The number of table entries to be set.
table_type	The type of lookup table entries to be set (see the Description).
entries	An array of table entries [§2.18.1, page 46].

Returns

None.

Description

Invoke **PEXSetTableEntries** to set the 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**

Errors

BadAlloc	The implementation 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, PEXGetDefinedIndices, PEXGetPredefinedEntries, PEXGetTableEntries, PEXGetTableEntry, PEXGetTableInfo

Chapter 11

Pipeline Contexts

A *pipeline context* is a PEX resource containing an instance of the set of attributes which describe a rendering pipeline state. Although many pipeline context resources may exist, at any particular time at most one is associated with a particular renderer. A pipeline context is associated with a renderer by setting the renderer's pipeline context attribute to the resource ID of the pipeline context. The attributes in the pipeline context associated with a renderer are used to initialize the renderer's pipeline state whenever one of the following functions are invoked: **PEXBeginPickAll**, **PEXBeginPickOne, PEXBeginRendering, PEXInitRenderer, PEXPickAll, PEXPickOne,** or **PEXRenderNetwork**. If no pipeline context is associated with a renderer, the renderer uses the default values listed below to initialize the pipeline state. The **PEXCopyPCToPipelineState** function differs slightly in that it modifies a renderer's pipeline state according to the contents of a pipeline context that is not necessarily the one associated with the renderer.

This pipeline context section describes pipeline context attributes and the operations that you can perform on pipeline context resources.

Some of the attributes in a pipeline context are references to other resources. If you create a resource and put its identifier in a pipeline context attribute, and then free that resource, then the implementation returns the value **PEXAlreadyFreed** as the resource identifier of the freed resource as the result of a query of that pipeline context attribute. The freed resource continues to exist as long as it continues to be referenced, either by the pipeline context or another resource.

If you wish to modify a reference to a resource to indicate that no resource is to be referenced, use the special resource identifier **None**.

Since each pipeline context attribute has a corresponding Output Command that modifies the same pipeline state attribute, look for the detailed description of the OC and pipeline context attribute in *Chapter 8:Output Attribute Functions*. In many cases, the data format used in the output command is very similar to the data format used in the pipeline context.

The following table contains the bitmask values you use with the **PEXSetPCAttributeMask** function, the member names of the corresponding values in the `PEXPCAttributes5_2` structure, and the default values that the pipeline context contains when you create one. Note that functions that use the `PEXPCAttributes` structure have two forms, one for PEXlib 5.1 and another for PEXlib 5.2. This is required because of changes in size of several data types used in the PEXPCAttributes structure. You must use the 5.2 form of the functions along with the 5.2 form of the data structure (`PEXPCAttributes5_2`) unless you are working entirely with data defined in PEX 5.0 or 5.1.

Table 20: Pipeline Context Member Names and Default Values

Bitmask Value	Member Name	Default Value	PEX
PEXPCActiveTextures	active_textures(page 385)	{}	(5.2)
PEXPCAll	all bits for items prior to PEX 5.2	imp-dep	(5.2)
PEXPCAll5_2	all bits for items defined up to and including PEX 5.2	imp-dep	(5.2)
PEXPCAlphaBlendFunction	alpha_blend_function (page 265)	**PEXAlphaBlendNone**	(5.2)

Table 20: Pipeline Context Member Names and Default Values (Continued)

Bitmask Value	Member Name	Default Value	PEX
PEXPCATextAlignment	atext_alignment (page 271)	{**PEXVAlignNormal**, **PEXHAlignNormal**}	(5.0)
PEXPCATextHeight	atext_height (page 273)	0.01	(5.0)
PEXPCATextPath	atext_path (page 274)	**PEXPathRight**	(5.0)
PEXPCATextStyle	atext_style (page 276)	**PEXATextNotConnected**	(5.0)
PEXPCATextUpVector	atext_up_vector (page 277)	<0.0, 1.0>	(5.0)
PEXPCASFValues	asf_enables, asf_values[a] (page 325)	{all selected, all **PEXIndivid-ual**}	(5.0)
PEXPCBFActiveTextures	bf_active_textures(page 292)	{ }	(5.2)
PEXPCBFDataMappingIndex	bf_data_mapping_index (page 277)	1	(5.2)
PEXPCBFDataMappingMethod	bf_data_mapping_method (page 279)	{**PEXDMMNone**, n/a}	(5.2)
PEXPCBFInteriorBundleIndex	bf_interior_bundle_index (page 281)	1	(5.2)
PEXPCBFInteriorStyle	bf_interior_style (page 283)	**PEXInteriorStyleHollow**	(5.0)
PEXPCBFInteriorStyleIndex	bf_interior_style_index (page 285)	1	(5.0)
PEXPCBFReflectanceBundleIndex	bf_reflectance_bundle_index (page 287)	1	(5.2)
PEXPCBFReflectionAttributes	bf_reflection_attr (page 288)	{1.0, 1.0, 1.0, 0.0, 0.0, {**PEXColorTypeIndexed**, 1}}	(5.0)
PEXPCBFRelectionModel	bf_reflection_model (page 290)	**PEXRMNone**	(5.0)
PEXPCBFSurfaceColor	bf_surface_color (page 294)	{**PEXColorTypeIndexed**, 1}	(5.0)
PEXPCBFSurfaceInterp	bf_surface_interp (page 297)	**PEXSurfaceInterpNone**	(5.0)
PEXPCCharExpansion	char_expansion (page 299)	1.0	(5.0)
PEXPCCharHeight	char_height (page 301)	0.01	(5.0)
PEXPCCharSpacing	char_spacing (page 302)	0.0	(5.0)
PEXPCCharUpVector	char_up_vector (page 303)	<0.0, 1.0>	(5.0)
PEXPCColorApproxIndex	color_approx_index (page 304)	0	(5.0)
PEXPCConditionFlags	condition_flags (page 424)	none set	(5.2)
PEXPCCullingMode	culling_mode (page 314)	**PEXNone**	(5.0)
PEXPCCurveApprox	curve_approx (page 305)	{1, n/a}	(5.0)

Table 20: Pipeline Context Member Names and Default Values (Continued)

Bitmask Value	Member Name	Default Value	PEX
PEXPCDataMappingIndex	data_mapping_index (page 307)	1	(5.2)
PEXPCDataMappingMethod	data_mapping_method (page 308)	{**PEXDMMNone**, n/a}	(5.2)
PEXPCDepthCueIndex	depth_cue_index (page 310)	0	(5.0)
PEXPCDistinguishFlag	distinguish (page 315)	**False**	(5.0)
PEXPCDrawingFunction	drawing_function (page 311)	*GXcopy*	(5.2)
PEXPCEdgeBundleIndex	edge_bundle_index (page 313)	1	(5.0)
PEXPCGlobalTransform	global_transform (page 316)	Identity	(5.0)
PEXPCGlobalTransformHints	global_transform_hints (page 316)	**PEXNone**	(5.2)
PEXPCHighlightIndex	highlight_index (page 322)	1	(5.2)
PEXPCHLHSRIdentifier	hlhsr_id (page 323)	**PEXHLHSRIDDisable**	(5.0)
PEXPCInteriorBundleIndex	interior_bundle_index (page 329)	1	(5.0)
PEXPCInteriorStyle	interior_style (page 331)	**PEXInteriorStyleHollow**	(5.0)
PEXPCInteriorStyleIndex	interior_style_index (page 333)	1	(5.0)
PEXPCLightState	light_state (page 335)	{ }	(5.0)
PEXPCLineBundleIndex	line_bundle_index (page 337)	1	(5.0)
PEXPCLineCapStyle	line_cap_style (page 338)	**PEXLineCapButt**	(5.2)
PEXPCLineColor	line_color (page 339)	{**PEXColorTypeIndexed**, 1}	(5.0)
PEXPCLineJoinStyle	line_join_style (page 342)	0 (imp-dep)	(5.2)
PEXPCLineMiterLimit	line_miter_limit (page 344)	10.0	(5.2)
PEXPCLineType	line_type (page 345)	**PEXLineTypeSolid**	(5.0)
PEXPCLineWidth	line_width (page 347)	1.0	(5.0)
PEXPCLocalTransform	local_transform (page 348)	Identity	(5.0)
PEXPCLocalTransformHints	local_transform_hints (page 348)	**None**	(5.2)
PEXPCMarkerBundleIndex	marker_bundle_index (page 352)	1	(5.0)
PEXPCMarkerColor	marker_color (page 353)	{**PEXColorTypeIndexed**, 1}	(5.0)
PEXPCMarkerScale	marker_scale (page 356)	1.0	(5.0)
PEXPCMarkerType	marker_type (page 357)	**PEXMarkerAsterisk**	(5.0)

Table 20: Pipeline Context Member Names and Default Values (Continued)

Bitmask Value	Member Name	Default Value	PEX
PEXPCModelClip	model_clip (page 359)	**PEXNoClip**	(5.0)
PEXPCModelClipVolume	model_clip_volume (page 360) (page 362)	{}	(5.0)
PEXPCMoreASFValues	more_asf_enables, more_asf_values[a] (page 325)	{all selected, all **PEXIndividual**}	(5.2)
PEXPCNameSet	name_set (page 262), (page 263)[b]	**None**	(5.0)
PEXPCParaSurfBundleIndex	para_surf_bundle_index (page 363)	1	(5.2)
PEXPCParaSurfChar	para_surf_char (page 364)	{**PEXPSCNone**, n/a}	(5.0)
PEXPCPatternRefPoint	pattern_ref_point (page 367) (page 369)	{0.0, 0.0, 0.0}	(5.0)
PEXPCPatternRefVec1	pattern_ref_vec1 (page 367) (page 369)	<1.0, 0.0, 0.0>	(5.0)
PEXPCPatternRefVec2	pattern_ref_vec2 (page 367) (page 369)	<0.0, 1.0, 0.0>	(5.0)
PEXPCPatternSize	pattern_size (page 370)	<1.0, 1.0>	(5.0)
PEXPCPickID	pick_id (page 372)	0	(5.0)
PEXPCPlaneMask	plane_mask (page 373)	**AllPlanes**	(5.2)
PEXPCPolylineInterp	polyline_interp (page 374)	**PEXPolylineInterpNone**	(5.0)
PEXPCPrimAntialiasing	primitive_antialiasing (page 376)	{imp-dep, 0 (imp-dep)}	(5.2)
PEXPCReflectanceBundleIndex	reflectance_bundle_index (page 378)	1	(5.2)
PEXPCReflectionAttr	reflection_attr (page 379)	{1.0, 1.0, 1.0, 0.0, 0,0, {**PEXColorTypeIndexed**, 1}}	(5.0)
PEXPCReflectionModel	reflection_model (page 381)	**PEXRMNone**	(5.0)
PEXPCRenderingColorModel	rendering_color_model (page 383)	0 (imp-dep)	(5.0)
PEXPCSequentialProcessing	sequential_processing (page 433)	**PEXOn**	(5.2)
PEXPCSurfaceApprox	surface_approx (page 387)	{1 (imp-dep), 1.0, 1.0}	(5.0)
PEXPCSurfaceColor	surface_color (page 390)	{**PEXColorTypeIndexed**, 1}	(5.0)
PEXPCSurfaceEdgeColor	surface_edge_color (page 393)	{**PEXColorTypeIndexed**, 1}	(5.0)
PEXPCSurfaceEdgeFlag	surface_edges (page 395)	**PEXOff**	(5.0)

Table 20: Pipeline Context Member Names and Default Values (Continued)

Bitmask Value	Member Name	Default Value	PEX
PEXPCSurfaceEdgeType	surface_edge_type (page 396)	**PEXSurfaceEdgeSolid**	(5.0)
PEXPCSurfaceEdgeWidth	surface_edge_width (page 398)	1.0	(5.0)
PEXPCSurfaceInterp	surface_interp (page 399)	**PEXSurfaceInterpNone**	(5.0)
PEXPCTextAlignment	text_alignment (page 401)	{**PEXVAlignNormal**, **PEXHAlignNormal**}	(5.0)
PEXPCTextBundleIndex	text_bundle_index (page 403)	1	(5.0)
PEXPCTextColor	text_color (page 404)	{**PEXIndexed**, 1}	(5.0)
PEXPCTextFont	text_font (page 407)	1	(5.0)
PEXPCTextPath	text_path (page 408)	**PEXPathRight**	(5.0)
PEXPCTextPrecision	text_precision (page 410)	**PEXStringPrecision**	(5.0)
PEXPCTMPerspectiveCorrection	tm_perspective_correction (page 412)	imp-dep	(5.2)
PEXPCTMResourceHints	tm_resource_hints (page 413)	{**PEXTMResourceHintNone**, {} }	(5.2)
PEXPCTMSampleFrequency	tm_sample_frequency (page 412)	**PEXTMSampleFrequencyPixel**	(5.2)
PEXPCVertexOrder	vertex_order (page 417)	**PEXRighthand**	(5.2)
PEXPCViewIndex	view_index (page 418)	0	(5.0)
PEXPCZBufferTest	z_buffer_test (page 419)	imp-dep	(5.2)

a. In a pipeline state, all aspect source flags are always enabled. Therefore the function **PEXGetPipelineContext** returns all of the aspect source flags via the *asf_values* and *more_asf_values* members of the PC attribute data structure.
b. When a pipeline state is initialized from a pipeline context, the contents of the name set are copied, not the resource identifier of the name set. In a pipeline state, the name set is stored as a list of names.

11.1. ASF Handling in the Pipeline Context for PEXlib 5.2

With the increased number of ASF's in PEX 5.2, two full words are required to store the value flags and two full words are required to store the enable flags in the pipeline context. Therefore, the bitmasks defined for PEXlib 5.1 in "PEXOCCSetIndividualASF - Set Individual Aspect Source Flags (5.0)" on page 325 are no longer adequate for setting ASF enable or value bits in PEXlib 5.2. There are two new ASF data members in the PC attributes data structure that you should use for PEXlib 5.2 applications:

```
PEXBitmask        more_asf_enables[2];
PEXBitmask        more_asf_values[2];
```

PEXlib 5.2 provides a new set of defined constants that represent the shift value for these bitmasks for each ASF. The algorithm for setting a bit for a given ASF is as follows:

```
pcattrs.more_asf_enables[X/32] |= 1L << X % 32;
```

where pcattrs is an instance of a `PEXPCAttributes` data structure and X is one of the constants defined below.

PEXASFPCMarkerType
PEXASFPCMarkerScale
PEXASFPCMarkerColor
PEXASFPCTextFontIndex
PEXASFPCTextPrec
PEXASFPCCharExpansion
PEXASFPCCharSpacing
PEXASFPCTextColor
PEXASFPCLineType
PEXASFPCLineWidth
PEXASFPCLineColor
PEXASFPCCurveApprox
PEXASFPCPolylineInterp
PEXASFPCInteriorStyle
PEXASFPCInteriorStyleIndex
PEXASFPCSurfaceColor
PEXASFPCSurfaceInterp
PEXASFPCReflectionModel
PEXASFPCReflectionAttr
PEXASFPCBFInteriorStyle
PEXASFPCBFInteriorStyleIndex
PEXASFPCBFSurfaceColor
PEXASFPCBFSurfaceInterp
PEXASFPCBFReflectionModel
PEXASFPCBFReflectionAttr
PEXASFPCSurfaceApprox
PEXASFPCSurfaceEdges
PEXASFPCSurfaceEdgeType
PEXASFPCSurfaceEdgeWidth
PEXASFPCSurfaceEdgeColor
PEXASFPCParaSurfCharacteristics
PEXASFPCDataMappingMethod
PEXASFPCBFDataMappingMethod
PEXASFPCLineCapStyle
PEXASFPCLineJoinStyle

PEXChangePipelineContext5_2 - Change Pipeline Context (5.0)

Synopsis

void **PEXChangePipelineContext5_2**(Display **display*, `PEXPipelineContext` *context*, unsigned long **value_mask*, `PEXPCAttributes5_2` **values*)

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 [§2.3.16, page 28].
value_mask	A pointer to an array of three unsigned long.
values	A pointer to new values for attributes in the pipeline context [§2.24.2, page 56], [§2.24.1, page 54].

Returns

None.

Description

Invoke **PEXChangePipelineContext5_2** to change the components of the specified pipeline context to the values you specify. You must use the 5.2 form of the function unless all aspects of the components you wish to change were defined in PEX 5.0 or 5.1. Note that each of the two forms uses a different structure for the *values* parameter.

Use the *value_mask* to indicate which attribute values you want changed. Invoke the **PEXSetPCAttributeMask** and **PEXSetPCAttributeMaskAll** functions to set up the *value_mask*.

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

PEXCreatePipelineContext5_2, **PEXGetPipelineContext5_2**, **PEXSetPCAttributeMask**, **PEXSetPCAttributeMaskAll**

PEXCopyPipelineContext - Copy Pipeline Context (5.0)

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 [§2.3.16, page 28].
dst_context	The resource identifier of the destination pipeline context [§2.3.16, page 28].

Returns

None.

Description

Invoke **PEXCopyPipelineContext** to copy attributes from the source pipeline context (*src_context*) to the destination pipeline context (*dst_context*).

Both pipeline contexts must already exist as valid pipeline context resources. This function copies only the attributes indicated by the *value_mask*. The remainder of the attributes are left as they were. Invoke the **PEXSetPCAttributeMask** and **PEXSetPCAttributeMaskAll** functions to set up the *value_mask*.

Errors

BadPEXPipelineContext	A specified pipeline context resource identifier is invalid.
BadValue	An invalid bit is set in the value mask.

See Also

PEXCreatePipelineContext5_2, **PEXSetPCAttributeMask**, **PEXSetPCAttributeMaskAll**

PEXCreatePipelineContext5_2 - Create Pipeline Context (5.0)

Synopsis

`PEXPipelineContext` **PEXCreatePipelineContext5_2**(Display **display*, unsigned long **value_mask*,
    `PEXPCAttributes5_2` **values*)

`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 [§2.24.2, page 56], [§2.24.1, page 54].

Returns

The resource identifier of the newly-created pipeline context [§2.3.16, page 28].

Description

Invoke **PEXCreatePipelineContext5_2** to create a pipeline context and obtain its resource identifier. You must use the 5.2 form of the function unless all aspects of the components you wish to supply were defined in PEX 5.0 or 5.1. Note that each of the two forms uses a different structure for the *values* parameter.

Use the *value_mask* to indicate the value you want to create. If you do not specify attribute values, then the implementation initializes the attribute values to the default values. Invoke the **PEXSetPCAttributeMask** and **PEXSetPCAttributeMaskAll** functions to set up the *value_mask*.

Errors

BadAlloc	The implementation 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, **PEXSetPCAttributeMask**, **PEXSetPCAttributeMaskAll**

PEXFreePCAttributes5_2 - Free Storage Returned by PEXGetPipelineContext (5.0)

Synopsis

void **PEXFreePCAttributes5_2**(`PEXPCAttributes5_2` **values*)

void **PEXFreePCAttributes**(`PEXPCAttributes` **values*)

Arguments

values	A pointer to the pipeline context attribute values [§2.24.2, page 56], [§2.24.1, page 54].

Returns

None.

Description

Invoke **PEXFreePCAttributes5_2** to deallocate memory returned by the **PEXGetPipelineContext5_2** function. Be sure to use the correct form of the function to match the type of the attributes data structure.

Errors

None.

See Also

PEXGetPipelineContext5_2

PEXFreePipelineContext - Free Pipeline Context (5.0)

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 [§2.3.16, page 28].

Returns

None.

Description

Invoke **PEXFreePipelineContext** to delete the association between the pipeline context resource identifier and the pipeline context.

The PEX implementation frees the pipeline context when no other resource references it.

Errors

BadPEXPipelineContext	The specified pipeline context resource identifier is invalid.

See Also

PEXCreatePipelineContext5_2

PEXGetPipelineContext5_2 - Get Pipeline Context Attributes (5.0)

Synopsis

`PEXPCAttributes5_2` ***PEXGetPipelineContext5_2**(Display **display*, `PEXPipelineContext` *context*, unsigned long **value_mask*)

`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 [§2.3.16, page 28].
value_mask	A pointer to an array of three unsigned long.

Returns

A pointer to pipeline context values [§2.24.2, page 56], [§2.24.1, page 54]; a null pointer if unsuccessful.

Description

Invoke **PEXGetPipelineContext5_2** to obtain the requested attribute values of the pipeline context. You must use the 5.2 form of the function unless all aspects of the components you wish to get were defined in PEX 5.0 or 5.1. Note that each of the two forms uses a different structure for the returned parameter.

Use the *value_mask* to indicate which attribute values you are requesting. Invoke the **PEXSetPCAttributeMask** and **PEXSetPCAttributeMaskAll** functions to set up the *value_mask*.

PEXlib allocates the memory for the return value. Invoke the **PEXFreePCAttributes5_2** function to deallocate the memory.

Errors

BadPEXPipelineContext	The specified pipeline context resource identifier is invalid.
BadValue	An invalid bit is set in the value mask.

See Also

PEXChangePipelineContext5_2, **PEXCreatePipelineContext5_2**, **PEXSetPCAttributeMask**, **PEXSetPCAttributeMaskAll**

PEXSetPCAttributeMask - Set Up Pipeline Context Attributes Value Mask (5.0)

Synopsis

void **PEXSetPCAttributeMask**(unsigned long **mask*, unsigned long *attr*)

Arguments

mask	The address of the value mask - an array of three unsigned long.
attr	A single pipeline context attribute bitmask constant.

Description

Invoke the **PEXSetPCAttributeMask** function to set up the bitmask for the pipeline context attributes. See "Pipeline Context Member Names and Default Values" on page 493 for the symbols to use in creating the bit-mask.

Errors

None.

See Also

PEXChangePipelineContext5_2, **PEXCopyPipelineContext**, **PEXCreatePipelineContext5_2**, **PEXGetPipelineContext5_2**, **PEXSetPCAttributeMaskAll**

PEXSetPCAttributeMaskAll - Set All Pipeline Context Attributes in Value Mask (5.0)

Synopsis

void **PEXSetPCAttributeMaskAll**(unsigned long **mask*)

Arguments

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

Description

Invoke the **PEXSetPCAttributeMaskAll** function to set up the bitmask for the pipeline context attributes. This function sets all valid bits in the mask.

Errors

None.

See Also

PEXChangePipelineContext5_2, **PEXCopyPipelineContext**, **PEXCreatePipelineContext5_2**, **PEXGetPipelineContext5_2**, **PEXSetPCAttributeMask**

Chapter 12

Renderers

A PEX *renderer* is the resource responsible for drawing output primitives and applying their attributes. You alter the behavior of the renderer by changing the renderer attributes. Some of these renderer attributes reference other resources, so you need to create these resources in order to have the renderer use them. These resources are: a pipeline context giving the initial values of the pipeline state, lookup tables holding values for many of the arguments used during rendering, and name sets identifying primitives to be highlighted, made invisible, or made detectable during picking. During rendering, you associate the renderer with one drawable, to which all output is directed. The renderer also supports picking on primitives. See "Renderer Picking" on page 561

Rendering Controls

Invoke the PEXlib output command functions (see "Programming Interface for Generating Output Commands" on page 133) to direct output commands directly to a renderer for *immediate rendering* or to a structure for *structure mode rendering*.

Immediate Rendering: With immediate rendering, the client program sends output commands directly to a renderer to be rendered immediately to the drawable. For immediate processing, start rendering by invoking the **PEXBeginRendering** function, which associates the renderer with a drawable. You can use the renderer in immediate mode by specifying **PEXOCRender** in the OC functions. You can associate a different drawable without changing the pipeline state by using the **PEXBindDrawableToRenderer** function. When finished rendering, invoke the **PEXEndRendering** function, which disassociates the renderer from the drawable.

Structure Mode Rendering: With structure mode rendering, you store output commands in structures (page 575). You can use the renderer in structure mode by specifying **PEXOCStore** in the OC functions and later instruct the renderer, with the **PEXRenderNetwork** function, to traverse these structure networks. The **PEXRenderNetwork** function implicitly does the operations done explicitly by the **PEXBeginRendering** and **PEXEndRendering** functions.

Mixed Mode Rendering: To render in mixed mode, invoke the **PEXExecuteStructure** function in immediate mode which traverses stored structures. If you want to traverse part of a stored structure in mixed mode, invoke the **PEXRenderElements** function.

A renderer is said to be "rendering" when its *renderer state* attribute is **PEXRendering** or **PEXMultipassRendering**.

Multipass Rendering

Multipass rendering enables the renderer to perform operations, such as multipass transparency, that require the output commands defining a single scene to be processed more than once. Specify the pipeline state and the renderer attributes to indicate the operations performed during multipass rendering to achieve the effects desired.

A multipass session is initiated by the **PEXInitMultipass** function. The renderer must already be in the **PEXRendering** state. All output commands subsequently passed to the renderer are considered part of a pass, many of which may be required to produce the requested effects. The first pass consists of the output commands sent to the renderer between the **PEXInitMultipass** function invocation and an invocation of the **PEXNextPass** function or **PEXNextPassWithoutReply** function. These latter two functions demarcate the end of the current pass and the beginning of the next. The implementation expects the same output commands to be sent in each pass.

One pass after another is performed until no more are needed. The number of passes required depends on the implementation and the types of operations you are performing. To determine the number of required passes, check the return value when you invoke the **PEXNextPass** function. This function returns a value indicating the number of additional passes needed, if the implementation can determine that, or simply that more passes are needed or that no more are needed.

A client performing simple multipass rendering begins a scene by invoking the **PEXInitMultipass** function. It then continually sends passes, demarcated by **PEXNextPass** function invocations, until the **PEXNextPass** function returns zero for the number of additional passes required. Options are defined for aborting a multipass session and for informing the implementation to produce the best image it can with only one more pass.

Whether an implementation is capable of multipass rendering is determined by corresponding support for capabilities that require multipass rendering. The only defined capabilities in PEX 5.2 that require multipass rendering are transparency methods *TransparencyTwoPass and TransparencyMultipass*.

Pipeline State

The *pipeline state* is an internal, opaque renderer data structure that maintains the values that attribute output commands can modify during rendering or picking. The pipeline state is created when the renderer is associated with a drawable by invoking the **PEXBeginRendering**, **PEXBeginPickOne**, or **PEXBeginPickAll** function and exists until the invocation of a **PEXEndRendering**, **PEXEndPickOne**, or **PEXEndPickAll** function. The pipeline state also exists for the duration of a **PEXRenderNetwork**, **PEXPickOne**, or **PEXPickAll** function.

The pipeline state contains the same information as a pipeline context, and is initialized by the pipeline context associated with the renderer during **PEXBeginRendering**, **PEXBeginPickOne**, **PEXBeginPickAll, PEXRenderNetwork**, and **PEXInitRenderer** functions.

Values in the pipeline state are modified by attribute output commands as well as **PEXCopyPCtoPipelineState**. The description of each output command defines which pipeline state value the output command modifies. When a renderer draws a primitive output command, the attribute values currently in effect in the pipeline state are applied to the primitive.

Renderer Interaction with Ancillary Buffers

A renderer manages the Z-buffer needed for some hidden line/hidden surface algorithms.

When an implicit or explicit **PEXBeginRendering** occurs and the renderer's *HLHSR_mode* indicates that Z-buffering should be performed or if the renderer's *clear Z* attribute is **True**, then the implementation obtains or allocates a Z-buffer suitable for use with the specified drawable, and binds it to the drawable. (A new Z-buffer is only allocated if a Z-buffer does not exist for the drawable or if the Z-buffer which is bound to the drawable is not suitable, e.g. a resize has occurred and the Z-buffer has not yet been resized.) If the renderer's *HLHSR_mode* does not indicate that Z-buffering is to be performed and the *clear Z* attribute is **False** at the time **PEXBeginRendering** is processed, but the mode is later changed by a **PEXChangeRendererExtAttributes** request to indicate that Z-buffering is to be performed, then the Z-buffer is allocated, if necessary, at that time.

When **PEXBeginRendering** is processed and if the renderer's *clear Z* attribute is **True**, then the Z-buffer is cleared to the renderer's *clear Z value* only in the region specified by the renderer's clip list. **PEXClearRenderer** performs the same operation if the Z clear parameter is on.

A renderer also manages the buffer used for alpha blending and certain methods of transparency. This buffer is known as an alpha buffer as it contains the resulting alpha values from certain blending operations. When an implicit or explicit **PEXBeginRendering** occurs and the renderer's *clear alpha* attribute is **True** or the renderer's *transparency method* indicates an alpha buffer is needed, an alpha buffer is allocated and bound to the drawable. If the *transparency method* does not indicate that an alpha buffer is needed when **PEXBeginRendering** is processed, but the method is later changed by a **PEXChangeRendererExtAttributes** request to a method which does require an alpha buffer, then the alpha buffer is allocated, if necessary, at that time. Also, if the transparency method is *TransparencyAlphaBlend*, then the alpha buffer is not allocated, if necessary, until the pipeline state's *alpha blend function* is set to a value which requires an alpha buffer (i.e., a function which uses destination alpha as an input).

When **PEXBeginRendering** is processed and if the renderer's *clear alpha* attribute is **True**, then the alpha buffer is cleared to the alpha value in the renderer's *background color*. If no alpha value is provided in the renderer's *background color*, then the default value of 1.0 is used. The alpha buffer is cleared only in the region specified by the renderer's clip list. **PEXClearRenderer** performs the same operation if the alpha clear parameter is on.

In addition to the conditions previously described, ancillary buffers can also be allocated by the implementation under certain circumstances in response to other renderer requests described in this chapter. For example, if a renderer is created and a **PEXClearRenderer** request to clear the Z buffer is received immediately after the renderer is created, the implementation allocates the Z buffer as a part of satisfying the request. Z buffers can be allocated as the result of the **PEXClearRenderer**, **PEXGetZBuffer**, **PEXPutZBuffer**, **PEXCopyZBufferToPixmap**, and **PEXCopyPixmapToZBuffer** functions. Alpha buffers can be allocated as the result of the **PEXClearRenderer**, **PEXCopyAlphaBufferToPixmap**, and **PEXCopyPixmapToAlphaBuffer** functions.

If the Z-buffer or alpha buffer could not be allocated, then the implementation issues an *BadAlloc* error and aborts rendering; the *renderer state* is set to **PEXIdle**.

Z-buffers and alpha buffers are not deallocated when the *renderer state* becomes **PEXIdle**. They remain bound to the drawable until you free the drawable. This allows the contents of the buffers to be reused if the corresponding clear flags are off. When the renderer state next becomes **PEXRendering** or **PEXPicking**, an attempt is made to reuse them. If they cannot be reused (for instance, if the drawable has been resized and the Z-buffer has not been resized to match), then they are reallocated as necessary.

Finally, a renderer also utilizes an accumulation buffer as another type of ancillary buffer. However, accumulation buffers are managed differently than other ancillary buffers. See *Chapter 17* for more information.

Related to this discussion on buffers is the function “PEXGetAncillaryBufferInfo - Get Ancillary Buffer Information (5.2)” on page 97 which provides information to determine the capabilities of various visuals such as whether Z-buffer, alpha buffer and accumulation buffers are supported for a specified drawable created with a particular visual.

Response to Drawable Changes

If the drawable associated with a renderer is destroyed or resized while the renderer state is **PEXRendering**, then an implicit **PEXEndRendering** is performed by the implementation with a *flush* argument of **False** in order to return the renderer to the **PEXIdle** state. All subsequent output and traversal commands are ignored. The implementation does not issue any errors; however, at a subsequent **PEXBeginRendering**, the implementation may issue a *BadAlloc* error if an attempt to reallocate ancillary buffers fails.

If the drawable associated with a renderer is moved, exposed, or occluded while the renderer state is **PEXRendering**, then the implementation continues processing subsequent requests using the newly-modified drawable.

The impact of drawable changes on the accumulation buffer is described in *Chapter 17*.

12.1. Renderer Attributes

The renderer attributes are listed in Table 21, “Renderer Attributes,” on page 510. They are set with the **PEXChangeRendererExtAttributes** function, and also can be specified when a renderer is created with a **PEXCreateRendererWithExtAttributes** function. The data structures associated with the renderer attributes are in “Data Structures for Renderer Attributes” on page 60.

You must use the “bitmask/value” interface to specify renderer attributes when using the **PEXChangeRendererExtAttributes, PEXCreateRendererWithExtAttributes**, or **PEXGetRendererExtAttributes** functions. You specify a bitmask that indicates which attributes you wish to set, change, or get. The size of the bitmask accepted by these functions is two words, as opposed to the single word masks used in older PEXlib functions. This change means that you can no longer pass the bitmask itself in these functions; you must pass a pointer to a multi-word bitmask. Note also, that the definition of the bitmask constants (the PEXRXA* symbols below) are no longer bitmasks, but are shift values. You should now use the **PEXSetRendererExtAttributeMask** function to set up bitmasks. You complete the “value” portion of the interface by putting the values you want to change in the `PEXRendererAttributes` data structure.

If a renderer attempts to read a value from a lookup table which has a resource ID of **None**(0), then the fallback attributes for the lookup table are used, except as noted in the individual descriptions of the lookup tables below. If a renderer attempts to read a value from a name set which has a resource ID of **None**, then the result is as if the name set exists but is empty. If the renderer's pipeline context attribute contains the resource ID of **None**, then the default values for the pipeline context are copied to the renderer's pipeline state during renderer initialization. If you wish to modify a reference to a resource to indicate that no resource is to be referenced, use the special resource identifier **None**.

Table 21: Renderer Attributes

Bitmask Value	Member Name	Default Value	PEX
PEXRXABackgroundColor	background_color	{**PEXColorTypeIndexed**, 0}	(5.1)
PEXRXAClearAlpha	clear_alpha	**False**	(5.2)
PEXRXAClearImage	clear_image	**False**	(5.1)
PEXRXAClearZ	clear_z	**True**	(5.1)
PEXRXAClearZValue	clear_z_value	0.0	(5.2)
PEXRXAClipList	clip_list	Null	(5.0)
PEXRXAColorApproxTable	color_approx_table	**None**	(5.0)
PEXRXAColorTable	color_table	**None**	(5.0)
PEXRXACurrentPath	current_path	Null	(5.0)
PEXRXADataMappingTable	data_mapping_table	**None**	(5.2)
PEXRXADepthCueTable	depth_cue_table	**None**	(5.0)
PEXRXAEchoColor	echo_color	implementation-dependent	(5.2)
PEXRXAEchoMethod	echo_method	implementation-dependent	(5.2)
PEXRXAEchoMode	echo_mode	**PEXNoEcho**	(5.1)
PEXRXAEdgeBundle	edge_bundle	**None**	(5.0)
PEXRXAExtentSizeTable	extent_size_table	**None**	(5.2)
PEXRXAHighlightExcl	highlight_excl	**None**	(5.0)
PEXRXAHighlightIncl	highlight_incl	**None**	(5.0)
PEXRXAHighlightTable	highlight_table	**None**	(5.2)
PEXRXAHLHSRMode	hlhsr_mode	**PEXOff**	(5.0)
PEXRXAInteriorBundle	interior_bundle	**None**	(5.0)
PEXRXAInvisibilityExcl	invisibility_excl	**None**	(5.0)
PEXRXAInvisibilityIncl	invisibility_incl	**None**	(5.0)
PEXRXALightTable	light_table	**None**	(5.0)
PEXRXALineBundle	line_bundle	**None**	(5.0)
PEXRXAMarkerBundle	marker_bundle	**None**	(5.0)
PEXRXAMarkerGlyphTable	marker_glyph_table	**None**	(5.2)
PEXRXANPCSubVolume	npc_subvolume	{(0.0,0.0,0.0),(1.0,1.0,1.0)}	(5.0)
PEXRXAPatternTable	pattern_table	**None**	(5.0)
PEXRXAPHIGSInteriorBundle	phigs_interior_bundle_mode	**False**	(5.2)
PEXRXAPickExcl	pick_excl	**None**	(5.1)
PEXRXAPickIncl	pick_incl	**None**	(5.1)

Table 21: Renderer Attributes (Continued)

Bitmask Value	Member Name	Default Value	PEX
PEXRXAPickStartPath	pick_start_path	Null	(5.1)
PEXRXAPipelineContext	pipeline_context	**None**	(5.0)
PEXRXAPixelColorMode	pixel_color_mode	**PEXOff**	(5.2)
PEXRXARendererState	renderer_state	**PEXIdle**	(5.0)
PEXRXATextBundle	text_bundle	**None**	(5.0)
PEXRXATextFontTable	text_font_table	**None**	(5.0)
PEXRXATextureMappingTable	texture_mapping_table	**None**	(5.2)
PEXRXATransparencyMethod	transparency_method	implementation-dependent	(5.2)
PEXRXATransparencyThresholds	transparency_thresholds	{1, 1.0}	(5.2)
PEXRXAViewport	viewport	{imp. dep., imp. dep., **True**}	(5.0)
PEXRXAViewTable	view_table	**None**	(5.0)

12.2. Renderer Attribute Descriptions

12.2.1. Background Color

Set the background color attribute to indicate the color to use when clearing the image buffer of the drawable associated with the renderer. The color approximation indicated by the pipeline state's *color approximation index* is applied to this color when the image buffer is cleared, and only the area identified by the renderer's *clip list* is given the background color. The alpha component of this background color is applied when the alpha buffer is cleared.

12.2.2. Clear Alpha

The clear alpha attribute indicates whether the alpha buffer, if one is bound to the current drawable, is cleared during renderer initialization. If you set *clear alpha* to **True**, then the alpha buffer is cleared to the alpha value specified in the renderer's *background color*, otherwise it is not. If the background color does not specify an alpha value, then the alpha value defaults to 1.0. Only pixels within the renderer's clip list are cleared. If you set *clear alpha* to **True**, and if the drawable associated with the renderer has the capability to support an alpha buffer, then an alpha buffer is allocated, if necessary.

12.2.3. Clear Image

The clear image attribute indicates whether the image buffer of the drawable associated with the renderer is cleared to the renderer's *background color* during renderer initialization. If you set *clear image* to **True**, then the image buffer is cleared; otherwise, the image buffer is not cleared. Only pixels within the renderer's *clip list* are modified during the clear.

12.2.4. Clear Z-buffer

The clear z-buffer attribute indicates whether the Z-buffer is cleared during renderer initialization. If you set *clear Z* to **True**, then the Z-buffer is cleared to the value specified in the renderer's *clear Z value*; otherwise, the Z-buffer is not cleared. Only pixels within the renderer's *clip list* are cleared. If you set *clear Z* to **True**, and if the drawable associated with the renderer has the capability to support a Z-buffer, then a Z-buffer is allocated, if necessary.

12.2.5. Clear Z-buffer Value

The clear Z value is in the range [0.0, 1.0] in DC space where 0.0 is the value farthest away from the viewer.

12.2.6. Clip List

The clip list is a list of rectangles in device coordinates that define the areas of the drawable in which rendering may occur. If the list is *Null*, then the entire drawable is eligible. If the list is not *Null*, then rendering occurs only in those pixels that are within the rectangles specified in the list. Overlapping rectangles in the clip list produce undefined results.

12.2.7. Color Approximation Table

The color approximation table identifies the color approximation table from which to draw the color approximation arguments indicated by the pipeline state's *color approximation index*. See "Color Approximation Table" on page 442

12.2.8. Color Table

The color table identifies the color table used to resolve indexed colors.

12.2.9. Current Path

The current path maintains the structure-network path formed by rendering requests. Set the path by invoking the **PEXBeginRendering**, **PEXRenderNetwork** and **PEXInitRenderer** functions. **PEXBeginRendering** and **PEXInitRenderer** set the path to a single element reference with a structure identifier of zero and an offset of zero. **PEXRenderNetwork** uses a network's root structure identifier and an offset of zero.

You cannot explicitly set this attribute. You can only inquire this attribute.

12.2.10. Data Mapping Table

The data mapping table attribute identifies the data mapping table from which to draw the data mapping methods indicated by the pipeline state's *data mapping index* and *data mapping back-facing index* and corresponding ASF's.

12.2.11. Depth Cue Table

The depth cue table attribute identifies the depth cue table from which to draw the depth cueing arguments indicated by the pipeline state's *depth cue index*.

12.2.12. Echo Color

The echo color attribute provides the color used by some echo methods when the *echo mode* renderer attribute is **PEXEcho** or **PEXUnEcho**. The echo method is determined by the *echo method* renderer attribute.

To support clients using PEX versions prior to 5.2, you can also modify *echo color* by invoking the escape **SetEchoColor**.

12.2.13. Echo Method

The echo method attribute indicates how primitives are to be echoed and unechoed. PEXlib defines the following echo methods:

> **PEXEchoDestructiveColor** - Echoes a primitive by drawing it in the color defined by the *echo color* renderer attribute. This color overrides the primitive's color. Lighting, shading, depth cueing, transparency, texture mapping, and data mapping are not applied to the primitive. The color is applied to both the front-facing and back-facing facets of surface primitives.
>
> This method is destructive, meaning the image rendered in echo mode **PEXEcho** cannot be redrawn correctly in echo mode **PEXUnEcho**. The image must be cleared and re-rendered to be restored. (5.2)
>
> **PEXEchoNonDestructiveColor** - This method is identical to method **PEXEchoDestructiveColor** except that the image is redrawn correctly in echo mode **PEXUnEcho**. (5.2)

PEXEchoXORColor - Echo a primitive by performing the logical XOR operation on the pixel values in the frame buffer that the primitive modifies with the pixel value obtained by applying the renderer's current color approximation to the renderer's *echo color* renderer attribute. The resulting pixels are stored back in the frame buffer, overriding the primitive's color. Lighting, shading, depth cueing, transparency, texture mapping, and data mapping are not applied to the primitive. The color is applied to both the front-facing and back-facing facets of surface primitives.

The pipeline state's *drawing function* is forced to behave like function *GXcopy* when echo method **PEXEchoXORColor** is enabled.

This method is non-destructive, meaning the image rendered in echo mode **PEXEcho** is redrawn correctly in echo mode **PEXUnEcho**. (5.2)

Whether primitives are to be echoed is controlled by the *echo mode* renderer attribute.

12.2.14. Echo Mode

The echo mode attribute indicates whether primitives are echoed, unechoed or drawn normally when passed to a renderer. The method used for echoing and unechoing is identified by the *echo method* renderer attribute.

If a primitive is designated to be both highlighted and echoed, then it is echoed but not highlighted.

The possible values are **PEXNoEcho** (the default), **PEXEcho**, and **PEXUnEcho**.

12.2.15. Edge Bundle Table

The edge bundle table attribute identifies the edge bundle table used to resolve bundled edge attributes. See “Edge Bundle Table” on page 447

12.2.16. Extent Size Table

The extent size table attribute identifies the extent size table from which to get the extent size used with the **PEXTestExtent** and **PEXTestExtent2D** output commands. See “Extent Size Table” on page 448

12.2.17. Highlight Exclusion Name Set

The highlight exclusion name set attribute identifies the name set to use to determine which primitives to not highlight.

12.2.18. Highlight Inclusion Name Set

The highlight inclusion name set attribute identifies the name set used to determine which primitives to highlight when they are rendered. Each primitive is highlighted that has one of its names, as given by the pipeline state's *name set*, in the renderer's *highlight inclusion name set* when the primitive is drawn, and none of its names in the renderer's *highlight exclusion name set*.

12.2.19. Highlight Table

The highlight table attribute identifies the highlight table from which to get the arguments which describe how to highlight primitives. An entry of the highlight table is selected by the pipeline state's *highlight index*. If no highlight table is associated with the renderer, then the method used for highlighting is implementation-dependent. See “Highlight Table” on page 449

12.2.20. HLHSR Mode

The HLHSR mode indicates the hidden line/hidden surface method to use during rendering. PEXlib defines the following HLHSR modes:

PEXHLHSROff - Draws all output primitives in the order they are processed. No attempt is made to remove hidden surfaces. The Z-buffer is not modified even if present. (5.0)

PEXHLHSRZBuffer - Resolves visibility at each pixel using a depth or Z-buffering technique. The Z-buffering method and the number of bits of precision in the z values is drawable-dependent. (5.0)

PEXHLHSRPainters - Buffers output primitives as they are processed. When an “end rendering” occurs with flush set to **True**, the primitives in the buffer are sorted based on the average depth and rendered back-to-front. (5.0)

PEXHLHSRScanline - Buffers output primitives as they are received. When an “end rendering” occurs with flush set to **True**, the primitives in the buffer are sorted and visibility is computed in scan line order. (5.0)

PEXHLHSRHiddenLineOnly - Draws only visible lines. Output primitives may be buffered as they are received. When an “end rendering” occurs with flush set to **True**, the primitives in the buffer are sorted and hidden line computation is performed. (5.0)

PEXHLHSRZBufferId -The same method as for Z-Buffer, but the pipeline state’s HLHSR identifier determines which primitives HLHSR applies to. (5.1)

12.2.21. Interior Bundle Table

The interior bundle table attribute identifies the interior bundle table used to resolve bundled interior attributes. See “Interior Bundle Table” on page 450

12.2.22. Invisibility Exclusion Name Set

The invisibility exclusion name set identifies the name set to use to determine which primitives are to be drawn (not to be invisible).

12.2.23. Invisibility Inclusion Name Set

The invisibility inclusion name set identifies the name set used to determine which primitives are to be invisible—not drawn—when passed to a renderer. Those primitives are invisible that have one of their names, as specified by the pipeline state’s *name set*, in the renderer’s *invisibility inclusion name set* when the primitive is passed to the renderer, and none of their names in the renderer’s *invisibility exclusion name set*.

12.2.24. Light Table

The light table attribute identifies the light table from which to draw light source definitions indicated by the pipeline state’s *light source state*. See “Light Table” on page 452

12.2.25. Line Bundle Table

The line bundle table attribute identifies the line bundle table used to resolve bundled line attributes. See “Line Bundle Table” on page 454

12.2.26. Marker Bundle Table

The marker bundle table attribute identifies the marker bundle table used to resolve bundled marker attributes. See “Marker Bundle Table” on page 455

12.2.27. Marker Glyph Table

The marker glyph table attribute identifies the marker glyph table from which to draw the definition of markers selected by the pipeline state’s *marker type*. If no marker glyph table is associated with the renderer, then the pipeline state’s *marker type* selects one of the registered marker types. See “Marker Glyph Table” on page 456

12.2.28. NPC Subvolume

The NPC subvolume attribute indicates the portion of normalized project coordinates that is to be mapped to the renderer's *viewport*.

12.2.29. Pattern Table

The pattern table attribute identifies the pattern table from which to draw pattern definitions. See "Pattern Table" on page 457

12.2.30. PHIGS Interior Bundle Mode

The PHIGS interior bundle mode attribute supports the interior bundle organization of PHIGS PLUS. See "Interior Bundle Table" on page 450

12.2.31. Pick Exclusion Name Set

The pick exclusion name set identifies the name set to use to determine which primitives are not to be detectable.

12.2.32. Pick Inclusion Name Set

The pick inclusion name set identifies the name set used to determine which primitives are detectable during picking. Detectable primitives are those that have one of their names, as specified by the pipeline state's *name set*, in the renderer's *pick inclusion name set* when the primitive is passed to the renderer, and none of their names in the renderer's *pick exclusion name set*. Other criteria described in the pick chapter (*Chapter 13*) are also used to determine detectable primitives.

12.2.33. Pick Start Path

The pick start path identifies the position within a structure network at which to begin the next **PEXPickAll** or **PEXBeginPickAll** search. The search begins at the structure element immediately following the one indicated by the *pick start path*. The pick start path is set by the **PEXPickAll** and **PEXEndPickAll** requests to the path of the last element processed during a pick traversal.

12.2.34. Pipeline Context

The requests **PEXBeginRendering**, **PEXInitRenderer**, and **PEXRenderNetwork** set all values in the renderer's pipeline state to the values in the pipeline context indicated by this attribute.

12.2.35. Pixel Color Mode

The pixel color mode attribute supports color defined as pixel values.

When the pixel color mode is **PEXOn**, if a color type is **PEXColorTypeIndexed**, then the index value is interpreted as a pixel value. The color lookup table is therefore not used to resolve **PEXColorTypeIndexed** colors. Colors of all other types are converted to pixel values by finding their closest match in the colormap of the drawable which is bound to the renderer. Table 22, "Behaviors Enabled by the Pixel Color Mode," on page 515 describes the values which override the given pipeline state and renderer attributes when the pixel color mode is **PEXOn**.

Table 22: Behaviors Enabled by the Pixel Color Mode

Attribute	Behavior
Pipeline state's *data map method ASF*	**PEXIndividual**
Pipeline state's *data map method*	None
Pipeline state's *bf data map method ASF*	**PEXIndividual**

Table 22: Behaviors Enabled by the Pixel Color Mode

Pipeline state’s *bf data map method*	None
Pipeline state’s *rendering color model*	ignored (rendering is done with pixels)
Pipeline state’s *vertex order*	ignored (no reflectance calculations)
Pipeline state’s *polyline interpolation method ASF*	**PEXIndividual**
Pipeline state’s *polyline interpolation method*	None
Pipeline state’s *surface interpolation method ASF*	**PEXIndividual**
Pipeline state’s *surface interpolation method*	None
Pipeline state’s *light source state*	empty
Pipeline state’s *active textures*	empty
Pipeline state’s *bf active textures*	empty
Renderer’s *depth cue table* entry’s *mode*	Disabled
Renderer’s *transparency method*	None
Pipeline state’s *primitive antialiasing*	no primitive types enabled
Renderer’s *color table*	ignored (indexed colors are pixel values)
Renderer’s *color approximation table*	ignored (color approximation is unnecessary)

To determine if your implementation supports pixel color mode, use *PixelColorSupported* with request **PEXGetImpDepConstants**.

12.2.36. Renderer State

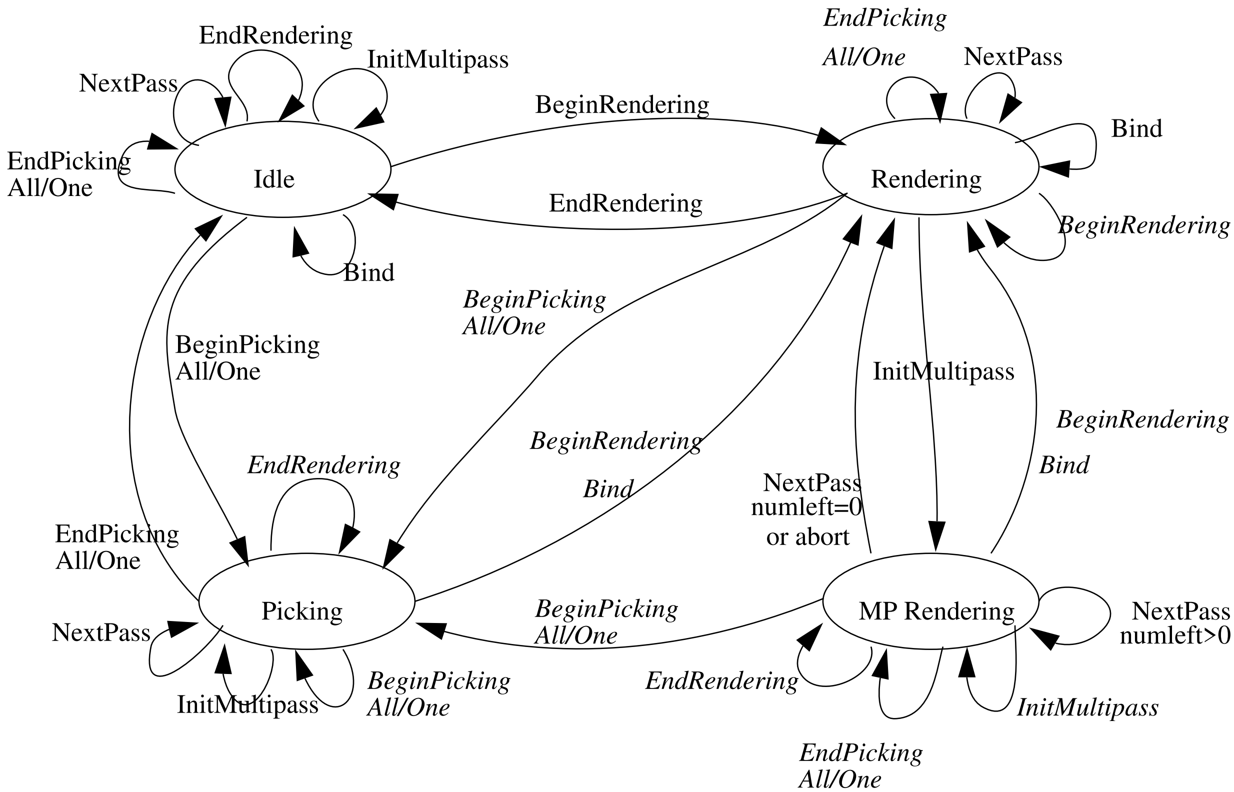
The renderer state attribute holds the current state of the renderer, either **PEXIdle**, **PEXRendering**, **PEXPicking** or **PEXMultipassRendering**. The state is set to **PEXRendering** by a **PEXBeginRendering** request and is subsequently set to **PEXIdle** by a **PEXEndRendering** request or certain error conditions (described in the requests that potentially raise those conditions). It also has the value **PEXRendering** while a **PEXRenderNetwork** request is in progress, but cannot be inquired during the traversal or processing of the request. The renderer state is set to **PEXPicking** by **PEXBeginPickOne** and **PEXBeginPickAll** requests, and subsequently to **PEXIdle** by **PEXEndPickOne** or **PEXEndPickAll** requests or certain error conditions. It also has the value **PEXPicking** during **PEXPickOne** or **PEXPickAll** requests. It is set to **PEXMultipassRendering** by **PEXInitMultipass**, and subsequently to **PEXRendering** by the final **PEXNextPass** or **PEXNextPassWithoutReply** which completes the multipass rendering.

You cannot explicitly set this attribute. You can only inquire this attribute.

The following figure illustrates a state transition diagram that shows the possible renderer state transitions. Each state is represented by an ellipse, with the directional lines indicating state transitions caused by various renderer functions. The requests illustrated in italics indicate transitions that generate renderer state errors.

The picking state has two “sub-states”, pick one and pick all. Renderer state errors also occur when mismatching these two sub-states, as when entering picking state with a **PEXBeginPickOne** request and then leaving with an **PEXEnd-PickAll** request. These sub-states, transitions, and errors are not shown, for clarity.

Renderer State Transitions



12.2.37. Text Bundle Table

The table bundle table attribute identifies the text bundle table used to resolve bundled text attributes. See “Text Bundle Table” on page 458

12.2.38. Text Font Table

The text font table attribute identifies the text font table from which to access the font group indicated by the pipeline state’s *text font index*. See “Text Font Table” on page 459

12.2.39. Texture Mapping Table

The texture mapping table identifies the texture mapping table from which to get texture map definitions indicated by the pipeline state’s *active textures* and *bf active textures*. See “Texture Mapping Table” on page 460

12.2.40. Transparency Method

The transparency method attribute indicates the method used to apply transparency. The alpha component of the primitive’s color is used for transparency computations. If the primitive does not define the alpha value, then the default value is determined differently depending on the primitive type. For surface primitives, the default alpha value is 1.0 - *transmission coefficient*. (The transmission coefficient is part of the reflection attributes.) For back-facing surfaces, when the pipeline state’s *facet distinguish flag* is **True**, the alpha value is 1.0 - *bf transmission coefficient*, where the back-facing transmission coefficient is defined by the pipeline state’s *bf reflection attributes*. For all other, non-surface primitives, the default alpha value is 1.0.

PEXlib defines the following transparency methods:

PEXTransparencyNone - No transparency. (5.2)

PEXTransparencyScreenDoor - Screen-door transparency. (5.2)

PEXTransparencyAlphaBlend - Alpha blending. If the implementation supports an alpha buffer, this mode can be used to perform transparency without multipass rendering. Otherwise, the client will have to sort primitives for reasonable results. The blending function used is determined by "PEXOCCSetAlphaBlend - Set Alpha Blend Function (5.2)" on page 265 (5.2)

PEXTransparencyTwoPass - Render only those primitives or portions of primitives with an alpha value of 1.0 in the first pass; render (blend) all other primitives in the second pass. (5.2)

PEXTransparencyMultipass - Meaningful for multipass rendering only; same as *transparency method None* if multipass rendering not active. The number of passes and the pixels drawn during each pass are controlled by the *transparency thresholds* renderer attribute. (5.2)

12.2.41. Transparency Thresholds

The transparency thresholds attribute defines the thresholds used by transparency method **PEXMultipassRendering**. The *transparency method* is also a renderer attribute. In the first pass, all pixels with alpha values greater or equal to the first threshold are rendered. In the i^{th} pass, where i is in the range [2..n], all pixels with alpha values less than the $(i-1)^{th}$ threshold value and greater than or equal to the i^{th} threshold value are rendered. In the $(n+1)^{th}$ pass, all pixels with alpha values less than the n^{th} threshold and greater than or equal to 0.0 are rendered. There are $n+1$ passes for n threshold values.

12.2.42. Viewport

The viewport is the area within a drawable in which the rendered image appears, specified in device coordinates (DC). A viewport which has boundaries outside of the drawable may be specified. Geometry that is mapped to values outside the drawable is clipped.

When the viewport is bound to the renderer, if the viewport's *use_drawable* flag is **True** the viewport width and height are obtained from the drawable's current width and height, and the viewport's depth is that indicated by the *viewport* attribute. If the *use_drawable* flag is **False**, then the viewport width, height and depth are all taken from the *viewport* attribute. The viewport is then set to the largest rectangle, justified to the lower left corner of the drawable, that achieves an isotropic mapping to the renderer's *NPC subvolume*. The viewport remains in the same position relative to the lower left hand corner of the drawable after a resize event occurs.

12.2.43. View Table

The view table attribute identifies the view table from which to draw the view definition indicated by the pipeline state's *view index*. See "View Table" on page 472

12.3. Renderer Requests

This section defines the requests that operate on renderers. Some requests defined by PEX versions 5.0 and 5.1 have been superseded. Those requests are defined in "Renderer Requests for PEXlib 5.0 and 5.1 Compatibility" on page 550.

PEXAccumulateState - Accumulate Rendering Pipeline State (5.1)

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 [§2.3.17, page 29].
count	The number of elements in the structure element reference path.
elements	A pointer to the structure element reference path [§2.26.5, page 62].

Returns

None.

Description

Invoke **PEXAccumulateState** to accumulate the pipeline state that would be in effect if a traversal were done to the element specified in the path.

If the renderer state is **PEXRendering** or **PEXPicking**, this function traverses to the structure element at the end of the list in *elements* and modifies only the pipeline state according to the output commands in that path. No primitives are rendered. The initial pipeline state is the one in effect at the time this function is invoked.

During the traversal, the *offset* member of the last entry in the renderer's *current path* is incremented by one for each structure element traversed, including those that have no effect on the pipeline state. No *structure* member of the *current path* is changed. No action beyond incrementing the *current path offset* is taken when Execute Structure, Conditional Execute Structure, or Conditional Return elements are traversed.

As with all paths, the non-leaf elements in the path must contain offsets that refer to structure referencing output commands. If the structure referenced by the leaf node is locked, then the offset must refer to element position 0 or a structure referencing output command.

If the renderer state is **PEXIdle**, then this function has no effect.

Errors

BadPEXPath	The specified path is invalid.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXRendererState	The renderer's state is **PEXMultipassRendering**.
BadPEXStructurePermission	The specified path includes an invalid position in a locked structure.

See Also

PEXBeginRendering, PEXCreateRendererWithExtAttributes, PEXCreateStructure, PEXRenderElements

PEXBeginRendering - Begin Rendering (5.0)

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 [§2.3.17, page 29].

Returns

None.

Description

Invoke **PEXBeginRendering** to initialize the specified renderer and to bind the specified drawable to it.

Subsequent output primitive commands sent to the renderer produce output to the specified drawable until the renderer receives a **PEXEndRendering** request or a **PEXBindDrawableToRenderer** request. 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 sets the renderer's state to **PEXRendering** and sets its current path to <0,0>. If the renderer's clear image attribute is **True**, then the renderer initializes the drawable's image buffer to the renderer's current background color within the areas defined by the clip list. If the renderer's clear Z attribute is **True**, then the renderer initializes the Z-buffer with the value in the clear Z value attribute and within the areas defined by the clip list. If the renderer's clear alpha attribute is **True**, then the renderer clears the alpha buffer to the values specified in the renderer's background color within areas defined by the clip list.

If the renderer's state is **PEXRendering, PEXPicking,** or **PEXMultipassRendering** when this request is received, then the implementation performs the above operations, but also generates an error. The new renderer state is then **PEXRendering**.

Errors

BadAlloc	The implementation 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 is in state **PEXRendering**, **PEXPicking**, **PEXMultipassRendering**.

See Also

PEXCreateRendererWithExtAttributes, PEXEndRendering

PEXBeginStructure - Save Rendering Pipeline State (5.0)

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 [§2.3.17, page 29].
structure_id	A value to be used as an application-specified structure identifier.

Returns

None.

Description

Invoke **PEXBeginStructure** to save the current values of the renderer's pipeline state.

If the renderer state of the renderer identified by *renderer* is **PEXRendering**, **PEXPicking**, or **PEXMultipass-Rendering**, then this function causes the renderer to save the current values of its pipeline state such that you can subsequently restore the values by a **PEXEndStructure** function. The renderer increments the offset field of the last entry in the renderer's current path and adds a new entry to the path. The renderer sets the structure field of this new entry to the value specified by *structure_id* and sets the offset field to zero. After saving the pipeline state, the renderer sets the pipeline state's global transform to the concatenation of the current local transform and the current global transform. The renderer then sets the local transform to the identity transform.

If the renderer state is **PEXIdle**, then this function has no effect.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.

See Also

PEXCreateRendererWithExtAttributes, PEXEndStructure, PEXExecuteStructure

PEXBindDrawableToRenderer - Bind Drawable to Renderer (5.2)

Synopsis

void **PEXBindDrawableToRenderer**(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 [§2.3.17, page 29].

Returns

None.

Description

Invoke **PEXBindDrawableToRenderer** to associate the specified *drawable* to the specified *renderer*.

The renderer directs its output to the specified drawable until the renderer receives another **PEXBindDrawableToRenderer** request or a **PEXEndRendering** request.

If the renderer's state is **PEXRendering** when this function is invoked, the renderer aborts the operation in progress with an implicit **PEXFlushRenderer** (with flush flag set to true) and remains in **PEXRendering** state. If the renderer's state is **PEXPicking** or **PEXMultipassRendering** when this function is invoked, then the renderer also performs the abort and flush operations, generates a renderer state error, and enters the **PEXRendering** state. If the renderer state is **PEXIdle** when this function is invoked, the renderer does nothing and does not issue an error.

Errors

BadAlloc	The implementation 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 state **PEXPicking** or **PEXMultipassRendering**.

See Also

PEXCreateRendererWithExtAttributes, PEXEndRendering

PEXChangeRendererExtAttributes - Change Renderer Extension Attributes (5.2)

Synopsis

void **PEXChangeRendererExtAttributes**(Display **display*, `PEXRenderer` *renderer*, unsigned long **value_mask*, `PEXRendererAttributes5_2` **values*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer [§2.3.17, page 29].
value_mask	A two word bitmask indicating the renderer attributes to be changed.
values	A pointer to new values for the renderer attributes [§2.26.2, page 61].

Returns

None.

Description

Invoke **PEXChangeRendererExtAttributes** to modify attributes of the specified renderer resource.

The value mask is a bitmask which can be manipulated with the **PEXSetRendererExtAttributeMask** function (page 548). See “Renderer Attributes” on page 510 for the bitmask names.

Attempts to set the renderer’s current path and renderer state attributes are ignored.

Errors

BadMatch	A specified lookup table resource was not created with a drawable compatible with the drawable used to create the renderer resource.
BadPEXLookupTable	A specified lookup table resource identifier is invalid.
BadPEXNameSet	A 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 in *values* is out of range, or an invalid bit is set in the value mask.

See Also

PEXCreateRendererWithExtAttributes, PEXGetRendererExtAttributes

PEXClearRenderer - Clear Renderer Buffers (5.2)

Synopsis

void **PEXClearRenderer**(Display **display*, `PEXRenderer` *renderer*, unsigned long *buffer_mask)*

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer [§2.3.17, page 29].
buffer_mask	A single word bitmask containing the logical OR of any of the following symbols: **PEXClearRendererImage**, **PEXClearRendererZ** or **PEXClearRendererAlpha**.

Returns

None.

Description

Invoke **PEXClearRenderer** to clear the buffers of the drawable associated with the renderer specified by *renderer* when the renderer is in **PEXRendering**, **PEXPicking**, or **PEXMultipassRendering** states.

PEXlib defines the following bits:

> **PEXClearRendererImage** - the renderer draws the renderer's background color into the drawable's image buffer in the regions indicated by the renderer's clip list.
>
> **PEXClearRendererZ** - the renderer clears the entire Z-buffer, if any, to the renderer's clear Z value in the regions indicated by the renderer's clip list.
>
> **PEXClearRendererAlpha** - the renderer draws the renderer's background color into the entire alpha buffer that is associated with the renderer, if any. Clear alpha buffer also respects the renderer's clip list. PEXlib uses the background color's *alpha component* to clear the alpha buffer. If there is no *alpha component*, then the alpha value defaults to 1.0.

If the renderer's state is **PEXIdle**, then this function has no effect.

Errors

BadAlloc	The implementation failed to allocate the alpha buffer or Z-buffer.
BadPEXRenderer	The specified renderer resource identifier is invalid.

See Also

PEXBeginRendering, PEXCreateRendererWithExtAttributes

PEXCopyAlphaBufferToPixmap - Copy Alpha Buffer to a Pixmap (5.2)

Synopsis

void **PEXCopyAlphaBufferToPixmap**(Display **display*, `PEXRenderer` *renderer*, Pixmap *pixmap)*

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer [§2.3.17, page 29].
pixmap	The resource identifier of the pixmap to receive the contents of the alpha buffer bound to the drawable associated with the renderer.

Returns

None.

Description

Invoke **PEXCopyAlphaBufferToPixmap** to copy the content of the alpha buffer to the *pixmap*.

If the renderer state attribute of the renderer specified by the *renderer* argument is **PEXRendering** or **PEXMultipassRendering**, then the renderer copies the content of the alpha buffer bound to the drawable associated with the renderer to the specified *pixmap*. The format of the values copied to the *pixmap* is implementation-dependent.

To determine the correct pixmap depth for copying the alpha buffer, see "PEXGetAncillaryBufferInfo - Get Ancillary Buffer Information (5.2)" on page 97.

The renderer's clip list is not considered by this function.

If the renderer state is **PEXPicking**, then the validity of the values copied from the alpha buffer is implementation-dependent.

If the renderer state is **PEXIdle**, then this function has no effect.

Errors

BadAlloc	The implementation failed to allocate the alpha buffer.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPixmap	The specified pixmap resource identifier is invalid.
BadMatch	The drawable associated with the renderer cannot have an alpha buffer.
BadMatch	The pixmap depth is not the same as the alpha buffer bound to the drawable associated with the renderer.
BadMatch	The pixmap's width or height do not match those of the drawable associated with the renderer.

See Also

PEXCopyPixmapToAlphaBuffer

PEXCopyPCToPipelineState - Copy PC to Pipeline State (5.2)

Synopsis

void **PEXCopyPCToPipelineState**(Display **display*, `PEXRenderer` *renderer*, `PEXPipelineContext` *pc*, unsigned long **mask)*

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer [§2.3.17, page 29].
pc	The resource identifier of the pipeline context, the values of which are copied into the renderer's pipeline state [§2.3.16, page 28].
mask	A pointer to a three word bitmask indicating which pipeline context values are copied.

Returns

None.

Description

Invoke **PEXCopyPCToPipelineState** to copy selected fields of the pipeline context.

If the renderer state of the renderer specified by the *renderer* argument is not **PEXIdle**, then the renderer copies selected fields of the pipeline context specified by *pc* to the renderer's pipeline state. The *mask* argument indicates which fields the renderer copies. Other fields in the renderer's pipeline state are not affected. Invoke the **PEXSetPCAttributeMask** and **PEXSetPCAttributeMaskAll** functions to set up the *mask*.

If the renderer state is **PEXIdle**, then this function has no effect.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXPipelineContext	The specified pipeline context resource identifier is invalid.
BadValue	An invalid bit is set in *mask*.

See Also

PEXCopyPipelineStateToPC

PEXCopyPipelineStateToPC - Copy Pipeline State To PC (5.2)

Synopsis

void **PEXCopyPipelineStateToPC**(Display **display*, `PEXRenderer` *renderer*, `PEXPipelineContext` *pc*, unsigned long **mask)*

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer [§2.3.17, page 29].
pc	The resource identifier of the pipeline context, into which the values of the renderer's pipeline state are copied [§2.3.16, page 28].
mask	A pointer to a three word bitmask indicating which pipeline state values are copied.

Returns

None.

Description

Invoke **PEXCopyPipelineStateToPC** to copy selected fields of the renderer's pipeline state.

If the renderer state of the renderer specified by the *renderer* argument is not **PEXIdle**, then the renderer copies selected fields of the renderer's pipeline state to the pipeline context specified by *pc*. The *mask* argument indicates which fields the renderer copies. Other fields in the specified pipeline context are not affected.

If the *mask* indicates that the pipeline state's current name set is to be copied, the pipeline context must contain the resource identifier of a valid name set resource into which the pipeline state's current name set values are copied. Any names existing in the name set resource prior to the copy operation are deleted.

Invoke the **PEXSetPCAttributeMask** and **PEXSetPCAttributeMaskAll** functions to set up the *mask*.

If the renderer state is **PEXIdle**, then this function has no effect.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXPipelineContext	The specified pipeline context resource identifier is invalid.
BadPEXNameSet	The name set resource ID of the pipeline context is invalid.
BadValue	An invalid bit is set in *mask*.

See Also

PEXCopyPCToPipelineState

PEXCopyPixmapToAlphaBuffer - Copy Pixmap to an Alpha Buffer (5.2)

Synopsis

void **PEXCopyPixmapToAlphaBuffer**(Display **display*, `PEXRenderer` *renderer*, Pixmap *pixmap)*

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer [§2.3.17, page 29].
pixmap	The resource identifier of the pixmap containing the alpha values to copy to the alpha buffer bound to the drawable associated with the renderer.

Returns

None.

Description

Invoke **PEXCopyPixmapToAlphaBuffer** to copy the content of the specified *pixmap* into the alpha buffer.

If the renderer state attribute of the renderer specified by the *renderer* argument is **PEXRendering** or **PEX-MultipassRendering**, then the renderer copies the content of the specified *pixmap* to the alpha buffer bound to the drawable associated with the renderer. The format of the values copied from the *pixmap* is implementation-dependent.

To determine the correct alpha buffer depth for copying the pixmap, see "PEXGetAncillaryBufferInfo - Get Ancillary Buffer Information (5.2)" on page 97

The renderer's clip list is not considered by this function.

If the renderer state is **PEXPicking**, then the values copied to the alpha buffer have an implementation-dependent effect.

If the renderer state is **PEXIdle**, then this function has no effect.

Errors

BadAlloc	The implementation failed to allocate the alpha buffer.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPixmap	The specified pixmap resource identifier is invalid.
BadMatch	The drawable associated with the renderer cannot have an alpha buffer.
BadMatch	The drawable depth is not the same as the alpha buffer bound to the drawable associated with the renderer.
BadMatch	The drawable's width or height do not match those of the drawable associated with the renderer.

See Also

PEXCopyAlphaBufferToPixmap

PEXCopyPixmapToZBuffer - Copy Pixmap to a Z-Buffer (5.2)

Synopsis

void **PEXCopyPixmapToZBuffer**(Display **display*, `PEXRenderer` *renderer*, Pixmap *pixmap)*

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer [§2.3.17, page 29].
pixmap	The resource identifier of the pixmap containing the Z values to copy to the Z-buffer bound to the drawable associated with the renderer.

Returns

None.

Description

Invoke **PEXCopyPixmapToZBuffer** to copy the content of the specified *pixmap* into the Z-buffer.

If the renderer state attribute of the renderer specified by the *renderer* argument is **PEXRendering** or **PEX-MultipassRendering**, then the renderer copies the content of the specified *pixmap* to the Z-buffer bound to the drawable associated with the renderer. The format of the values copied from the *pixmap* is implementation-dependent.

To determine the correct Z-buffer depth for copying the pixmap, see “PEXGetAncillaryBufferInfo - Get Ancillary Buffer Information (5.2)” on page 97.

The renderer’s clip list is not considered by this function.

If the renderer state is **PEXPicking**, then the values copied to the Z-buffer have an implementation-dependent effect.

If the renderer state is **PEXIdle**, then this function has no effect.

Errors

BadAlloc	The implementation failed to allocate the Z-buffer.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPixmap	The specified pixmap resource identifier is invalid.
BadMatch	The drawable associated with the renderer cannot have a Z-buffer.
BadMatch	The drawable depth is not the same as the Z-buffer bound to the drawable associated with the renderer.
BadMatch	The drawable’s width or height do not match those of the drawable associated with the renderer.

See Also

PEXCopyZBufferToPixmap

PEXCopyZBufferToPixmap - Copy Z-Buffer to a Pixmap (5.2)

Synopsis

void **PEXCopyZBufferToPixmap**(Display **display*, `PEXRenderer` *renderer*, Pixmap *pixmap)*

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer [§2.3.17, page 29].
pixmap	The resource identifier of the pixmap to receive the contents from the Z-buffer bound to the drawable associated with the renderer.

Returns

None.

Description

Invoke **PEXCopyZBufferToPixmap** to copy the content of the Z-buffer to the specified *pixmap*.

If the renderer state attribute of the renderer specified by the *renderer* argument is **PEXRendering** or **PEXMultipassRendering**, then the renderer copies the content of the Z-buffer bound to the drawable associated with the renderer to the specified pixmap. The format of the values copied to the pixmap is implementation-dependent.

To determine the correct pixmap depth for copying the Z-buffer, see "PEXGetAncillaryBufferInfo - Get Ancillary Buffer Information (5.2)" on page 97.

The renderer's clip list is not considered by this function.

If the renderer state is **PEXPicking**, then the validity of the values copied from the Z-buffer is implementation-dependent.

If the renderer state is **PEXIdle**, then this function has no effect.

Errors

BadAlloc	The implementation failed to allocate the Z-buffer.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPixmap	The specified pixmap resource identifier is invalid.
BadMatch	The drawable associated with the renderer cannot have a Z-buffer.
BadMatch	The drawable depth is not the same as the Z-buffer bound to the drawable associated with the renderer.
BadMatch	The drawable's width or height do not match those of the drawable associated with the renderer.

See Also

PEXCopyPixmapToZBuffer

PEXCreateRendererWithExtAttributes - Create Renderer With Extended Attributes (5.2)

Synopsis

`PEXRenderer` **PEXCreateRendererWithExtAttributes**(Display **display*, Drawable *drawable*, unsigned long **value_mask*, `PEXRendererAttributes5_2` **values*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable. The renderer will render to drawables which has a root, depth, and visual identifier that are the same as those of this example drawable.
value_mask	A two word bitmask indicating which renderer attribute values are specified by *values*.
values	A pointer to values used to override default values in the new renderer resource [§2.26.2, page 61].

Returns

The resource identifier of the newly-created renderer resource [§2.3.17, page 29].

Description

Invoke **PEXCreateRendererWithExtAttributes** to create a renderer and get the identifier of the newly-created renderer resource.

You may use this renderer to render to drawables with the same root, depth, and visual ID as the drawable specified by *drawable*. The *value_mask* is a bitmask which can be manipulated with the **PEXSetRendererExtAttributeMask** function (page 548). See "Renderer Attributes" on page 510 for the bitmask names.

Renderer attributes not explicitly set have default values. See "Renderer Attributes" on page 510

Attempts to set the renderer's current path and renderer state attributes are ignored.

Errors

BadAlloc	The implementation failed to allocate the resource.
BadDrawable	The specified drawable resource identifier is invalid.
BadMatch	The specified drawable is unsupported, or one of the specified lookup table resources was not created for the same root, depth and visual ID as the specified drawable.
BadPEXLookupTable	A specified lookup table resource identifier is invalid.
BadPEXNameSet	A specified name set resource identifier is invalid.
BadPEXPipelineContext	The specified pipeline context resource identifier is invalid.
BadValue	A specified value in *values* is out of range, or an invalid bit is set in the value mask.

See Also

PEXFreeRenderer

PEXEndRendering - End Rendering (5.0)

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 [§2.3.17, page 29].
flush	Specifies whether any pending output for *renderer* is to be rendered onto its associated drawable (**True** or **False**).

Returns

None.

Description

Invoke **PEXEndRendering** to terminate rendering on the specified renderer resource.

If you set *flush* to **True**, then pending output is rendered onto its associated drawable. If you set *flush* to **False**, then pending output is discarded. In either case, the renderer state is set to **PEXIdle**.

If the renderer state is **PEXIdle**, then the implementation ignores this request. If the renderer state is **PEXPicking** or **PEXMultipassRendering**, then the implementation issues a *BadPEXRendererState* error and does not change the renderer's state.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXRendererState	The specified renderer's state is **PEXPicking** or **PEXMultipassRendering**.

See Also

PEXBeginRendering, PEXCreateRendererWithExtAttributes

PEXEndStructure - Restore Rendering Pipeline State (5.0)

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 [§2.3.17, page 29].

Returns

None.

Description

Invoke **PEXEndStructure** to restore the previously saved rendering pipeline state.

If the renderer state of the renderer specified by *renderer* is **PEXRendering**, **PEXPicking**, or **PEXMultipassRendering**, then this function replaces the values in the renderer's pipeline state with those of the last-saved copy of the pipeline state (saved during the most recent **PEXBeginStructure** function). The last entry of the renderer's current path is removed. The renderer increments the offset of the new last entry of the current path as it processes subsequent output commands.

If the renderer state is **PEXIdle**, then this function has no effect.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXRendererState	No matching **PEXBeginStructure** function was invoked by the client prior to invoking this function.

See Also

PEXBeginStructure, PEXCreateRendererWithExtAttributes, PEXExecuteStructure

PEXFlushRenderer - Flush Renderer (5.2)

Synopsis

void **PEXFlushRenderer**(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 the renderer [§2.3.17, page 29].
flush	**True** if all pending output is rendered onto the drawable associated with the renderer. **False** to discard all pending output.

Returns

None.

Description

Invoke **PEXFlushRenderer** to force all pending output to be either drawn or discarded.

If you set *flush* to **True**, then the renderer renders all pending output onto the drawable associated with the renderer. If you set *flush* to **False**, then the renderer discards all pending output.

If the renderer state is **PEXIdle**, then this function has no effect.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.

See Also

PEXEndRendering

PEXFreeRenderer - Free Renderer (5.0)

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 [§2.3.17, page 29].

Returns

None.

Description

Invoke **PEXFreeRenderer** to delete the renderer resource specified by *renderer* and frees the memory associated with it. If the renderer's state is **PEXRendering** or **PEXMultipassRendering**, then an implicit **PEXEndRendering** request with the flush flag **False** is performed before the renderer is freed. If the renderer's state is **PEXPicking**, then the pick operation is aborted before the renderer is freed.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.

See Also

PEXCreateRendererWithExtAttributes

PEXFreeRendererExtAttributes - Free Storage Returned by PEXGetRendererExtAttributes (5.2)

Synopsis

void **PEXFreeRendererExtAttributes**(`PEXRendererAttributes5_2` **values*)

Arguments

values	A pointer to the renderer attribute values [§2.26.2, page 61].

Returns

None.

Description

Invoke **PEXFreeRendererExtAttributes** to deallocate memory returned by **PEXGetRendererExtAttributes**.

Errors

None.

See Also

PEXGetRendererExtAttributes

PEXGetRendererExtAttributes - Get Renderer Extended Attribute Values (5.2)

Synopsis

`PEXRendererAttributes5_2` ***PEXGetRendererExtAttributes**(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 [§2.3.17, page 29].
value_mask	A two word bitmask indicating attributes to be returned from the renderer.

Returns

A pointer to the renderer attribute values [§2.26.2, page 61]; a null pointer if unsuccessful.

Description

Invoke **PEXGetRendererExtAttributes** to get attribute values from the specified renderer resource.

The *value mask* is a bitmask which can be manipulated with the **PEXSetRendererExtAttributeMask** function (page 548). See “Renderer Attributes” on page 510 for the bitmask names.

PEXlib allocates the memory for the returned renderer attribute values. Invoke the **PEXFreeRendererExtAttributes** function to deallocate the memory.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadValue	An invalid bit is set in the value mask.

See Also

PEXChangeRendererExtAttributes, PEXCreateRendererWithExtAttributes, PEXFreeRendererExtAttributes

PEXGetZBuffer - Get Z-Buffer (5.2)

Synopsis

`PEXPointer` **PEXGetZBuffer**(Display **display*, `PEXRenderer` *renderer*, int *x*, int *y*, unsigned int *width*, unsigned int *height*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer of which the Z-buffer is to be queried [§2.3.17, page 29].
x,y	The DC origin of the Z-buffer rectangle to be queried.
width	The DC size (width) forming the Z-buffer rectangle to be queried.
height	The DC size (height) forming the Z-buffer rectangle to be queried.

Returns

A pointer to the Z-buffer values; a null pointer if unsuccessful.

Description

Invoke **PEXGetZBuffer** to get the content of the Z-buffer.

If the renderer state attribute of the renderer specified by *renderer* is **PEXRendering** or **PEXMultipassRendering**, then the implementation returns a pointer to the content of the Z-buffer described by the given rectangle. The validity of the values returned if the renderer state is **PEXPicking** is implementation-dependent. If the renderer state is **PEXIdle**, then the implementation returns a null pointer.

The *x* and *y* coordinates, specified in device coordinates, are relative to the renderer's current drawable origin and define the upper left corner of the rectangle. The *width* and *height*, specified in pixels, are inclusive. The function returns a null pointer if the rectangle is not contained within the drawable. The returned data is undefined for any part of the rectangle that is obscured. The renderer does not consider the clip list when executing this function.

The function returns the values row by row, starting with the upper left of the rectangle. The implementation maps the device specific Z buffer values into the range of [0.0, 1.0] in DC space and the returned values are of type float.

PEXlib allocates the memory for the returned Z-buffer values. Invoke **XFree** to deallocate the memory.

To determine implementation specific information about the Z-buffer, see "PEXGetAncillaryBufferInfo - Get Ancillary Buffer Information (5.2)" on page 97.

Errors

BadAlloc	The implementation failed to allocate a Z-buffer.
BadMatch	The drawable associated with the renderer cannot have a Z-buffer.
BadPEXRenderer	The specified renderer resource identifier is invalid.

See Also

PEXPutZBuffer

PEXInitMultipass - Initialize Multipass Rendering (5.2)

Synopsis

void **PEXInitMultipass**(Display **display*, `PEXRenderer` *renderer*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource of the identifier to be put into the **PEXMultipassRendering** state [§2.3.17, page 29].

Returns

None.

Description

Invoke **PEXInitMultipass** to initialize multipass rendering.

If the renderer state of the renderer specified by *renderer* is **PEXRendering**, then the renderer changes state to **PEXMultipassRendering** and initializes itself for multipass rendering. The first pass is defined as all functions and OC's targeted to this renderer subsequent to **PEXInitMultipass** and prior to **PEXNextPass** or **PEXNextPassWithoutReply**. Subsequent passes are defined as function invocations between sequential invocations of **PEXNextPass** or **PEXNextPassWithoutReply**.

If the renderer state is already **PEXMultipassRendering** when you invoke this function, then the renderer performs the operations listed above and issues a *BadPEXRendererState* error.

Support for multipass rendering is implementation-dependent. You can determine the level of support by querying the level of support of facilities that require multipass rendering, such as transparency methods.

If the renderer state is **PEXIdle** or **PEXPicking** or if the implementation does not support multipass rendering, then this function has no effect.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXRendererState	The renderer state is already **PEXMultipassRendering**

See Also

PEXNextPass, PEXNextPassWithoutReply

PEXInitRenderer - Initialize Renderer (5.2)

Synopsis

void **PEXInitRenderer**(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 [§2.3.17, page 29].

Returns

None.

Description

Invoke **PEXInitRenderer** to set the pipeline state of the renderer specified by *renderer* to the values contained in the pipeline context associated with the renderer.

Any saved pipeline state values (e.g., from a **PEXBeginStructure** function) are lost. If no pipeline context is associated with the renderer, then the renderer uses the default pipeline context values. The renderer also sets the current path attribute to a single element with a structure identifier of zero and an offset of zero.

If the renderer's renderer state is **PEXIdle**, then this request has no effect.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.

See Also

PEXBeginRendering, PEXCreateRendererWithExtAttributes

PEXNextPass - Next Pass (Multipass Rendering) (5.2)

Synopsis

int **PEXNextPass**(Display **display*, `PEXRenderer` *renderer*, int *next_pass*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer [§2.3.17, page 29].
next_pass	Specifies when the implementation should complete processing (**PEXMultipassContinue, PEXMultipassLastPass** or **PEXMultipassAbort).**

Returns

Number of passes required to complete multipass operation.

Description

Invoke **PEXNextPass** to signify the end of a multipass-rendering pass for the renderer identified by *renderer*, and, if applicable, the beginning of the next pass for that renderer.

The renderer executes any processing necessary to finish the pass. The number of additional passes required, if known, is returned by this function.

PEXlib defines the following values for *next_pass*:

PEXMultipassContinue - the implementation prepares to receive the next pass, which the client is expected to send unless the return value is 0.

PEXMultipassLastPass - the implementation prepares to complete the multipass operation with only one more pass. The implementation produces the best output it can using the multipass algorithms it is currently using for the multipass session. The client is expected to send the additional pass unless the return value is 0.

PEXMultipassAbort - the implementation terminates the multipass session. The effect on the rendered image is implementation-dependent. The *renderer state* is changed to **PEXRendering**. The number of additional passes that were required, if known, for this multipass session are returned by the function.

If the client sends an additional pass when that pass was not required, then the result is implementation-dependent.

Except where defined otherwise in this description, the returned value indicates whether additional passes are needed to complete the current multipass session. A value of zero indicates that rendering is complete and that this request caused the renderer state to become **PEXRendering**. A value of −1 indicates that the number of passes needed to complete rendering is unknown, and the client is expected to send passes until a count of zero is returned.

When a multipass session ends, whether fully completed or prematurely terminated, all resources other than the frame buffer, such as the Z-buffer and accumulation buffers, are left in an implementation-dependent state. Therefore, continuing in single-pass rendering mode after completing multipass typically requires initialization of all applicable states and resources.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadValue	The *next_pass* argument contains an invalid value.

See Also

PEXInitMultipass, **PEXNextPassWithoutReply**

PEXNextPassWithoutReply - Next Pass Without Reply (Multipass Rendering) (5.2)

Synopsis

void **PEXNextPassWithoutReply**(Display **display*, `PEXRenderer` *renderer*, int *next_pass*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer [§2.3.17, page 29].
next_pass	Specifies when the implementation should complete processing (**PEXMultipassContinue, PEXMultipassLastPass** or **PEXMultipassAbort).**

Returns

None.

Description

Invoke **PEXNextPassWithoutReply** to signify the end of a multipass-rendering pass for the renderer identified by *renderer*, and, if applicable, the beginning of the next pass for that renderer.

This function behaves the same as **PEXNextPass** with the single exception that there is no return value.

This function is useful in cases where the application already knows how many passes are left to go, perhaps from information received from a previous invocation of **PEXNextPass**. It may be more efficient because round trips to the implementation are not required for the replies.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadValue	The *next_pass* argument contains an invalid value.

See Also

PEXInitMultipass, **PEXNextPass**

PEXPutZBuffer - Put Z-Buffer (5.2)

Synopsis

void **PEXPutZBuffer**(Display **display*, `PEXRenderer` *renderer*, int *x*, int *y*, unsigned int *width*, unsigned int *height*, `PEXPointer` *values*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer [§2.3.17, page 29].
x,y	The DC origin of the Z-buffer rectangle to be loaded.
width	The DC size (width) forming the Z-buffer rectangle to be loaded.
height	The DC size (height) forming the Z-buffer rectangle to be loaded.
values	A pointer to the values to store in the Z-buffer.

Returns

None.

Description

Invoke **PEXPutZBuffer** to write the specified values to the Z-buffer.

If the renderer state attribute of the renderer specified by *renderer* is **PEXRendering** or **PEXMultipassRendering**, then the implementation writes the supplied values to the described rectangle of the Z-buffer. If the renderer state is **PEXPicking**, the values written have an implementation-dependent effect. If the renderer state is **PEXIdle**, then nothing is written to the Z-buffer.

The *x* and *y* coordinates, specified in device coordinates, are relative to the renderer's current drawable origin and define the upper left corner of the rectangle. The *width* and *height,* specified in pixels*,* are inclusive. Nothing is written to regions outside of the drawable or to obscured regions of the drawable. The renderer does not consider the clip list when executing this function.

The values are ordered row by row, starting with the upper left of the rectangle. The Z-buffer values are specified in the range of [0.0, 1.0] in DC space and are mapped to device specific values by the implementation. The values are of type float.

To determine implementation specific information about the Z-buffer, see "PEXGetAncillaryBufferInfo - Get Ancillary Buffer Information (5.2)" on page 97.

Errors

BadAlloc	The implementation failed to allocate the Z-buffer.
BadMatch	The drawable associated with the renderer cannot have a Z-buffer.
BadPEXRenderer	The specified renderer resource identifier is invalid.

See Also

PEXGetZBuffer

PEXRenderElements - Render Elements (5.1)

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 [§2.3.17, page 29].
structure	The resource identifier of a structure resource [§2.3.19, page 29].
whence1	A value specifying, with *offset1*, the first limit of the element range (**PEXBeginning**, **PEXCurrent** or **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** or **PEXEnd**).
offset2	An offset from *whence2* specifying the second limit of the element range.

Returns

None.

Description

Invoke **PEXRenderElements** to process all output commands in the specified element range of the specified *structure*.

The output primitives in the specified range of *structure* are rendered using the specified *renderer*. If the renderer state is **PEXIdle**, then the implementation ignores this request. Structures referenced through 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. PEXlib defines the following whence values:

> **PEXBeginning** - the element position is the value of offset (i.e., the offset from the beginning of the structure).
>
> **PEXCurrent** - 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** - 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, the offset is less than zero, then the position is set to zero. If after computing an element position, the offset is greater than the number of elements in the structure, then the position is set to the last structure element in the structure.

For locked structures, the whence/offset pairs must specify the element position 0 or the offset of a structure referencing output command.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.

BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The beginning or end of the specified element range denote an invalid position in a locked structure.
BadValue	The specified value for *whence* was invalid.

See Also

PEXBeginRendering, PEXCreateRendererWithExtAttributes, PEXCreateStructure, PEXAccumulateState

PEXRenderNetwork - Render Network (5.0)

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 [§2.3.17, page 29].
structure	The resource identifier of a structure resource [§2.3.19, page 29].

Returns

None.

Description

Invoke **PEXRenderNetwork** to process all output commands stored in *structure* using the specified *renderer*.

The output primitives in the structure are rendered to the specified *drawable*. Structures referenced by structure elements are also processed. This request effectively performs an implicit **PEXBeginRendering** before the traversal with an initial current path of (*structure*,0) and an implicit **PEXEndRendering**, with a *flush* value of **True**, after the traversal.

If the renderer's state is **PEXRendering**, **PEXPicking**, or **PEXMultipassRendering** when this request is received, then the implementation performs the above operations, but also generates an error.

Errors

BadAlloc	The implementation 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 is not in renderer state **PEXIdle**.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXBeginRendering, PEXCreateRendererWithExtAttributes, PEXCreateStructure, PEXEndRendering, PEXExecuteStructure

PEXSetRendererExtAttributeMask - Set Up Renderer Extended Attribute Value Mask (5.2)

Synopsis

PEXSetRendererExtAttributeMask(unsigned long **mask*, unsigned long *attr*)

Arguments

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

Description

Invoke the **PEXSetRendererExtAttributeMask** to aid in setting up the bitmask for the renderer attributes. You must use the bitmask constants in Table 21, "Renderer Attributes," on page 510.

Errors

None.

See Also

PEXChangeRendererExtAttributes, PEXCreateRendererWithExtAttributes, PEXGetRendererExtAttributes, PEXSetRendererExtAttributeMaskAll

PEXSetRendererExtAttributeMaskAll - Set All Renderer Extended Attribute Value Masks (5.2)

Synopsis

PEXSetRendererExtAttributeMaskAll(unsigned long **mask*)

Arguments

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

Description

Invoke the **PEXSetRendererExtAttributeMaskAll** function to set all the bits in the bitmask for all defined renderer attributes.

Errors

None.

See Also

PEXChangeRendererExtAttributes, PEXCreateRendererWithExtAttributes, PEXGetRendererExtAttributes, PEXSetRendererExtAttributeMask

12.4. Renderer Requests for PEXlib 5.0 and 5.1 Compatibility

The functions in this section are retained for compatibility with older versions of PEXlib. They only support the PEX 5.1 renderer attributes; you must use the newer function set to change or inquire attributes added in PEX 5.2.

The compatibility functions include:

PEXChangeRenderer - superseded by **PEXChangeRendererExtAttributes** (page 523).

PEXCreateRenderer - superseded by **PEXCreateRendererWithExtAttributes** (page 531).

PEXFreeRendererAttributes - superseded by **PEXFreeRendererExtAttributes** (page 536).

PEXGetRendererAttributes - superseded by **PEXGetRendererExtAttributes** (page 537).

PEXGetRendererDynamics - superseded by no function, since all renderer attributes in PEX 5.2 are considered dynamic.

PEXChangeRenderer - Change Renderer Attributes (5.0)

Note: This function has been superseded by the **PEXChangeRendererExtAttributes** function (page 523), but it is maintained here for backwards compatibility with PEXlib 5.1.

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 [§2.3.17, page 29].
value_mask	A mask indicating the renderer attributes to be changed.
values	A pointer to new values for the renderer attributes [§2.26.1, page 60].

Returns

None.

Description

Invoke **PEXChangeRenderer** to modify 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 some of 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

You cannot explicitly set the renderer's current path and renderer state attributes using this function.

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 (5.0)

Note: This function has been superseded by the **PEXCreateRendererWithExtAttributes** function (page 531), but it is maintained here for backwards compatibility with PEXlib 5.1.

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 [§2.26.1, page 60].

Returns

The resource identifier of the newly-created renderer resource [§2.3.17, page 29].

Description

Invoke **PEXCreateRenderer** to create 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 some of 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). You can use the renderer resource only in conjunction with drawables that have the same root and depth as specified drawable.

You cannot explicitly set the renderer's current path and renderer state attributes using this function.

Errors

BadAlloc	The implementation 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

PEXFreeRendererAttributes - Free Storage Returned by PEXGetRendererAttributes (5.0)

Note: This function has been superseded by the **PEXFreeRendererExtAttributes** function (page 536), but it is maintained here for backwards compatibility with PEXlib 5.1.

Synopsis

void **PEXFreeRendererAttributes**(`PEXRendererAttributes` **values*)

Arguments

values	A pointer to the renderer attribute values [§2.26.1, page 60].

Returns

None.

Description

Invoke **PEXFreeRendererAttributes** to deallocate memory returned by **PEXGetRendererAttributes**.

Errors

None.

See Also

PEXGetRendererAttributes

PEXGetRendererAttributes - Get Renderer Attribute Values (5.0)

Note: This function has been superseded by the **PEXGetRendererExtAttributes** function (page 537), but it is maintained here for backwards compatibility with PEXlib 5.1.

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 [§2.3.17, page 29].
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 [§2.26.1, page 60].

Description

Invoke **PEXGetRendererAttributes** to get the 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 some of 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. Invoke **PEXFreeRendererAttributes** to deallocate the memory.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadValue	An invalid bit is set in the value mask.

See Also

PEXChangeRenderer, **PEXCreateRenderer**, **PEXFreeRendererAttributes**, **PEXGetRendererDynamics**,

PEXGetRendererDynamics - Get Renderer Attribute Modification Dynamics (5.0)

Note: This function is not useful with PEX 5.2 implementations because all renderer attributes are defined to be dynamic in PEX 5.2. The function is maintained here for applications that work with PEX 5.0 and 5.1 implementations.

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 [§2.3.17, page 29].
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

Invoke **PEXGetRendererDynamics** to get 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. PEXlib defines the following bit values:

PEXDynamic - indicates that the specified attribute may be modified at any time, and the change will take place immediately.

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

PEXBeginRendering, PEXChangeRenderer, **PEXCreateRenderer**

Chapter 13

Renderer Picking

The semantics of renderer picking are identical to rendering except that during picking, primitives are hit tested instead of converted 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 - returns, at most, one primitive which satisfies the pick criteria.

pick all - returns 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. The pick inclusion and pick exclusion filters control which primitives are to be tested.

13.1. Pick Device Types

To inquire the supported pick device types, invoke the **PEXGetEnumTypeInfo** function. PEXlib defines the following pick types:

PEXPickDeviceDCHitBox - This type 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, or circular). The pick distance defines the half-width or radius of the pick area.

PEXPickDeviceNPCHitVolume - This type is a parallelepiped defined by its two opposite corners in normalized projection coordinates. Primitives that lie within or intersect the pick area or volume are considered “hit”.

13.2. Pick Method

The pick method determines which “hit” primitives are picked. For *pick one*, PEXlib defines four standard pick methods:

PEXPickLast - Returns the last “hit” primitive which satisfies the pick filter test.

PEXPickClosestZ - Returns the “hit” primitive which has a *z* value closest to the front clipping plane and satisfies the pick filter test. If multiple primitives satisfy this criteria, some of them may be returned.

PEXPickVisibleAny - Returns any “hit” primitive which is visible (after HLHSR) and satisfies the pick filter test.

PEXPickVisibleClosest - All “hit” primitives which are visible are selected. Of those selected, the one which is closest to the pick position and satisfies the pick filter test is returned. If you specify an NPC pick volume, then the pick position is the center of the volume.

The Z-buffer contents may not be preserved when using the *visible* pick methods. The *visible* pick methods only return primitives which would be visible in the pick aperture if they were rendered using the renderer's current HLHSR mode. For example, if the HLHSR mode is enabled, then 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.

For *pick all* (multiple primitives may be picked), PEXlib defines two standard pick methods:

> **PEXPickAllAll** - Returns the primitives, up to the maximum number of hits, which lie within or intersect the hit box.
>
> **PEXPickAllVisible -** Returns the primitives visible (after HLHSR), up to the maximum number of hits, and that lie within or intersect the hit box.

A "pick first" can be accomplished by specifying the maximum number of hits to be one.

To inquire the supported pick methods, invoke the **PEXGetEnumTypeInfo** function.

13.3. Pick Inclusion and Exclusion Filters

The renderer's pick inclusion and exclusion filters specify the name sets to be used to filter primitives during picking. See "Renderer Attributes" on page 510. The pick functions only return primitives which are not excluded by the pick filter. *pick one* returns 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.

13.4. Pick Start Path

In addition to pick device type and pick inclusion and exclusion filters, you can further control *pick all* by using 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.

13.5. Drawables Associated With Renderers

If a drawable that is associated with a renderer is destroyed or resized while the renderer is performing an immediate mode *pick one* or *pick all* hit test, then the pick operation is terminated and the renderer state is set to **PEXIdle.** The implementation ignores all subsequent output and traversal requests to the renderer until the next **PEXEndPickOne** or **PEXEndPickAll** (whichever is appropriate) which returns 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, then the implementation continues 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 (5.1)

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 [§2.3.17, page 29].
structure_id	A value to be used as an application-specified structure identifier.
method	The pick all method (**PEXPickAllAll** or **PEXPickAllVisible**).
send_event	Specifies whether the implementation should send an event when the maximum number of hits is reached (**True** or **False**).
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 [§2.25.8, page 60].

Returns

None.

Description

Invoke **PEXBeginPickAll** to start 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 the maximum hits specified. Additional picked primitives are not recorded. Once the maximum number of hits is reached, then the implementation may ignore subsequent primitives.

Inquire the supported pick device types by invoking the **PEXGetEnumTypeInfo** function.

The specified structure identifier is inserted as the first structure component in the returned pick path(s).

PEXlib defines two standard pick methods:

> **PEXPickAllAll** - Returns the primitives, up to the maximum number of hits, which lie within or intersect the hit box.
>
> **PEXPickAllVisible -** Returns the primitives visible (after HLHSR), up to the maximum number of hits, and that lie within or intersect the hit box.

If the send event flag is **True** and the pick method is **PEXPickAllAll**, then a **PEXMaxHitsReached** event is sent from the implementation to the client whenever implementation reaches the specified maximum number of hits, if the implementation supports the event. To determine the maximum number of supported hits, invoke the **PEXGetImpDepConstants** function. Upon receiving the event, the application should stop sending primitives and process the recorded hits. If the pick method is **PEXPickAllVisible**, then a complete set of primitives must be sent to the implementation before the implementation can determine which primitives to select.

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, then the implementation issues a *BadMatch* error. To determine the supported drawables, invoke the **PEXMatchRenderingTargets** function**.** If the renderer state is set to **PEXRendering**, **PEXMultipassRendering** or **PEXPicking** when this function is called, then the operation in progress is aborted, the **PEXBeginPickAll** function is completed, and the implementation issues a *BadPEXRendererState* error.

When the renderer state is **PEXPicking**, you can invoke any function which processes output commands or manipulates attributes (e.g., all output command functions, **PEXBeginStructure**, **PEXEndStructure**, **PEXRenderElements**, and **PEXAccumulateState**). These functions have the same semantics except that primitives are hit tested instead of converted to pixels.

Errors

BadAlloc	The implementation 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 not **PEXIdle**.
BadValue	The pick record contains invalid data, or the pick device type is invalid.

See Also

PEXEndPickAll, **PEXGetImpDepConstants**, **PEXPickAll**

PEXBeginPickOne - Begin Renderer Pick One (5.1)

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 [§2.3.17, page 29].
structure_id	A value to be used as an application-specified structure identifier.
method	The pick one method (**PEXPickLast**, **PEXPickClosestZ**, **PEXPickVisibleAny** or **PEXPickVisibleClosest**).
pick_device_type	The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).
pick_record	A pointer to the pick data record [§2.25.8, page 60].

Returns

None.

Description

Invoke **PEXBeginPickOne** to start 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. For *pick one*, a hierarchical path to the picked primitive is maintained.

PEXlib defines the following pick one methods:

PEXPickLast - Returns the last "hit" primitive which satisfies the pick filter test.

PEXPickClosestZ - Returns the "hit" primitive which has a z value closest to the front clipping plane and satisfies the pick filter test. If multiple primitives satisfy this criteria, any of them may be returned.

PEXPickVisibleAny - Returns any "hit" primitive which is visible (taking the current HLHSR mode into account) and satisfies the pick filter test.

PEXPickVisibleClosest - All "hit" primitives which are visible are selected. Of those selected, the one which is closest to the pick position and satisfies the pick filter test is returned. If you specify an NPC pick volume, then the pick position is the center of the volume.

To determine the supported pick device types, invoke the **PEXGetEnumTypeInfo** function.

The specified structure identifier is 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 implementation does not support the specified drawable, then the implementation issues a *BadMatch* error. To determine the supported drawables, invoke the **PEXMatchRenderingTargets** function. If the renderer state is set to **PEXRendering** or **PEXPicking** when you invoke this function, then the operation in progress is aborted, the **PEXBeginPickOne** function is completed, and the implementation issues a

BadPEXRendererState error.

When the renderer state is **PEXPicking**, you can invoke any function which processes output commands or manipulates attributes (e.g., all output command functions, **PEXBeginStructure**, **PEXEndStructure**, **PEXRenderElements**, and **PEXAccumulateState**). These functions have the same semantics except that primitives are hit tested instead of converted to pixels.

Errors

BadAlloc	The implementation 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 not **PEXIdle**.
BadValue	The pick record contains invalid data, or the pick device type is invalid.

See Also

PEXEndPickOne, **PEXPickOne**

PEXEndPickAll - End Pick All (5.1)

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 [§2.3.17, page 29].
status_return	Returns the status of the pick operation (**PEXPick**, **PEXNoPick** or **PEXAbortedPick**).
more_return	Returns the status of remaining picks (**PEXNoMoreHits**, **PEXMoreHits** or **PEXMayBeMoreHits**).
count_return	Returns the number of pick paths.

Returns

An array of pick paths [§2.25.7, page 60]; a null pointer if unsuccessful or no pick (see also *status_return*).

Description

Invoke **PEXEndPickAll** to terminate an immediate-mode pick, get the hierarchical pick paths of any hit primitives, and set the renderer state to **PEXIdle**.

If one or more primitives were picked, then this function returns a pick status of **PEXPick** 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, then 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, then the returned pick status is **PEXAbortedPick** and the returned pick paths is a null pointer.

The element positions of picked primitives in locked structures are undefined.

If all hits were recorded, then **PEXNoMoreHits** is returned and the renderer's pick start path is 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 is 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 picked element is in a locked structure the pick start path is set to the first structure referencing output command after the picked primitive, or to the output command referencing the locked structure if there are no subsequent structure referencing output commands in the locked structure.

If the renderer state is **PEXIdle** when you invoke this function (i.e., no picking is in progress or the rendering was aborted due to a resize), then the implementation ignores this function and does not issue any errors. If the renderer state is currently **PEXRendering** or **PEXMultipassRendering**, or if the pick operation in progress is a *pick one*, then the implementation issues a *BadPEXRendererState* error and does not change the renderer's renderer state.

PEXlib allocates memory for the return value. Invoke **PEXFreePickPaths** to deallocate the memory.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXRendererState	The renderer state is **PEXRendering** or **PEXMultipassRendering**, or was set to **PEXPicking** through function **PEXBeginPickOne**.

See Also

PEXBeginPickAll, **PEXFreePickPaths, PEXPickAll**

PEXEndPickOne - End Pick One (5.1)

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 [§2.3.17, page 29].
status_return	Returns the status of the pick operation (**PEXPick**, **PEXNoPick** or **PEXAbortedPick**).
undetectable_return	Returns **True** if another primitive better satisfies the pick criteria with the exception that it did not pass the pick filter test.

Returns

A pointer to the pick path [§2.25.7, page 60]; a null pointer if unsuccessful or no pick (see also *status_return*).

Description

Invoke PEXEndPickOne to terminate an immediate-mode pick, get the hierarchical pick path to the closest or last hit primitive, and set the renderer state to **PEXIdle**.

If a primitive was picked, then the returned pick status is **PEXPick**. If no primitive was picked, then 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, then 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 is **True**. Otherwise, it is **False**.

If the renderer state is currently **PEXIdle** when you invoke this function (i.e., no picking is in progress or the rendering was aborted due to a resize), then the implementation ignores this function and does not issue any errors. If the renderer state is currently **PEXRendering** or **PEXMultipassRendering**, or if the pick operation in progress is a *pick all*, then the implementation issues a *BadPEXRendererState* error and does not change the renderer's renderer state.

PEXlib allocates memory for the return value. Invoke **PEXFreePickPaths** to deallocate the memory.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXRendererState	The renderer state is **PEXRendering** or **PEXMultipassRendering**, or was set to **PEXPicking** through function **PEXBeginPickAll**.

See Also

PEXBeginPickOne, **PEXPickOne**

PEXFreePickPaths - Free Memory Allocated for Pick Paths (5.1)

Synopsis

void **PEXFreePickPaths**(unsigned long *count*, `PEXPickPath` **pick_paths*)

Arguments

count	The number of pick paths.
pick_paths	An array of pick paths [§2.25.7, page 60].

Returns

None.

Description

Invoke **PEXFreePickPath** to deallocate memory returned by **PEXEndPickAll, PEXEndPickOne, PEXPickAll** and **PEXPickOne**.

Errors

None.

See Also

PEXEndPickAll, **PEXEndPickOne, PEXPickAll**. **PEXPickOne**

PEXPickAll - Pick All Traversal (5.1)

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 [§2.3.17, page 29].
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 [§2.25.8, page 60].
status_return	Returns the status of the pick operation (**PEXPick**, **PEXNoPick** or **PEXAbortedPick**).
more_return	Returns the status of remaining picks (**PEXNoMoreHits**, **PEXMoreHits** or **PEXMayBeMoreHits**).
count_return	Returns the number of pick paths.

Returns

An array of pick paths [§2.25.7, page 60]; a null pointer if unsuccessful or no pick (see also *status_return*).

Description

Invoke **PEXPickAll** to traverse 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 specified 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, then the implementation issues a *BadPEXPath* error and returns a null pick path. Invoke **PEXCreateRendererWithExtAttributes** or **PEXChangeRendererWithExtAttributes** to initialize the renderer's current pick start path.

PEXlib defines the following *pick all* methods:

> **PEXPickAllAll** - Returns the primitives, up the maximum number of hits, which lie within or intersect the hit box.
>
> **PEXPickAllVisible -** Returns the primitives visible (taking into account the current HLHSR mode), up to the maximum number of hits, and that lie within or intersect the hit box.

Inquire the supported pick device types by invoking the **PEXGetEnumTypeInfo** function.

If one or more primitives were picked, then 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, then 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, then the returned pick status is **PEXAbortedPick** and the returned pick paths is a null pointer.

The element positions of picked primitives in locked structures are undefined.

The paths of all hit primitives are recorded until reaching the maximum number of hits or until the implementation maximum number of recordable hits is reached. Once the maximum number of paths is recorded, the implementation may ignore subsequent primitives and return the results.

If all possible hits were recorded, then **PEXNoMoreHits** is returned and the renderer's pick start path is 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 is 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 picked element is in a locked structure the pick start path is set to the first structure referencing output command after the picked primitive, or to the output command referencing the locked structure if there are no subsequent structure referencing output commands in the locked structure.

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, then the implementation issues a *BadMatch* error. To determine the supported drawables, invoke the **PEXMatchRenderingTargets** function. If the renderer state is set to **PEXRendering** or **PEXPicking** when you invoke this function, then the operation in progress is aborted, the **PEXPickAll** function is completed, and the implementation issues a *BadPEXRendererState* error.

PEXlib allocates memory for the return value. Invoke **PEXFreePickPaths** to deallocate the memory.

Errors

BadAlloc	The implementation 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 renderer state was not **PEXIdle**.
BadPEXPath	The renderer pick start path is invalid.
BadPEXStructurePermission	The renderer pick start path includes an invalid position in a locked structure.
BadValue	The pick record contains invalid data, or the pick device type is invalid.

See Also

PEXBeginPickAll, **PEXEndPickAll**, **PEXFreePickPaths**

PEXPickOne - Pick One Traversal (5.1)

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 [§2.3.17, page 29].
structure	The resource identifier for the root structure of the structure network [§2.3.19, page 29].
method	The pick one method (**PEXPickLast**, **PEXPickClosestZ**, **PEXPickVisibleAny** or **PEXPickVisibleClosest**).
pick_device_type	The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).
pick_record	A pointer to the pick data record [§2.25.8, page 60].
status_return	Returns the status of the pick operation (**PEXPick**, **PEXNoPick** or **PEXAbortedPick**).
undetectable_return	Returns **True** if another primitive better satisfies the pick criteria with the exception that it did not pass the pick filter test.

Returns

A pointer to the pick path [§2.25.7, page 60]; a null pointer if unsuccessful or no pick (see also *status_return*).

Description

Invoke **PEXPickOne** to traverse the specified structure network.

PEXlib defines the following *pick one* methods:

> **PEXPickLast** - Returns the last “hit” primitive which satisfies the pick filter test.
>
> **PEXPickClosestZ** - Returns the “hit” primitive which has a *z* value closest to the front clipping plane and satisfies the pick filter test. If multiple primitives satisfy this criteria, any of them may be returned.
>
> **PEXPickVisibleAny** - Returns some “hit” primitive which is visible (taking the current HLHSR mode into account) and satisfies the pick filter test.
>
> **PEXPickVisibleClosest** - All “hit” primitives which are visible are selected. Of those selected, the one which is closest to the pick position and satisfies the pick filter test is returned. If you specify an NPC pick volume, then the pick position is the center of the volume.

Inquire the supported pick device types by invoking the **PEXGetEnumTypeInfo** function.

If a primitive was picked, then the returned pick status is **PEXPick**. If no primitive was picked, then 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, then 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 is **True**. Otherwise, it is **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, then the implementation issues a *BadMatch* error. To determine the supported drawables, invoke the **PEXMatchRenderingTargets** function. If the renderer state is set to **PEXRendering** or **PEXPicking** when you invoke this function, then the operation in progress is aborted, the **PEXPickOne** function is completed, and the implementation issues a *BadPEXRendererState* error.

PEXlib allocates memory for the return value. Invoke **PEXFreePickPaths** to deallocate the memory.

Errors

BadAlloc	The implementation 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 renderer state was not **PEXIdle**.
BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The pick record contains invalid data, or the pick device type is invalid.

See Also

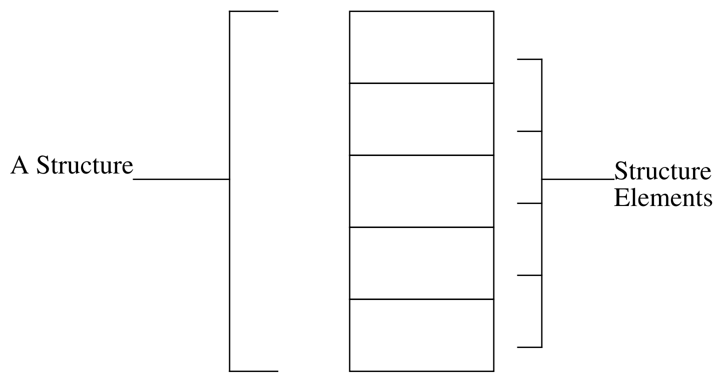
PEXBeginPickOne, **PEXEndPickOne**

Chapter 14

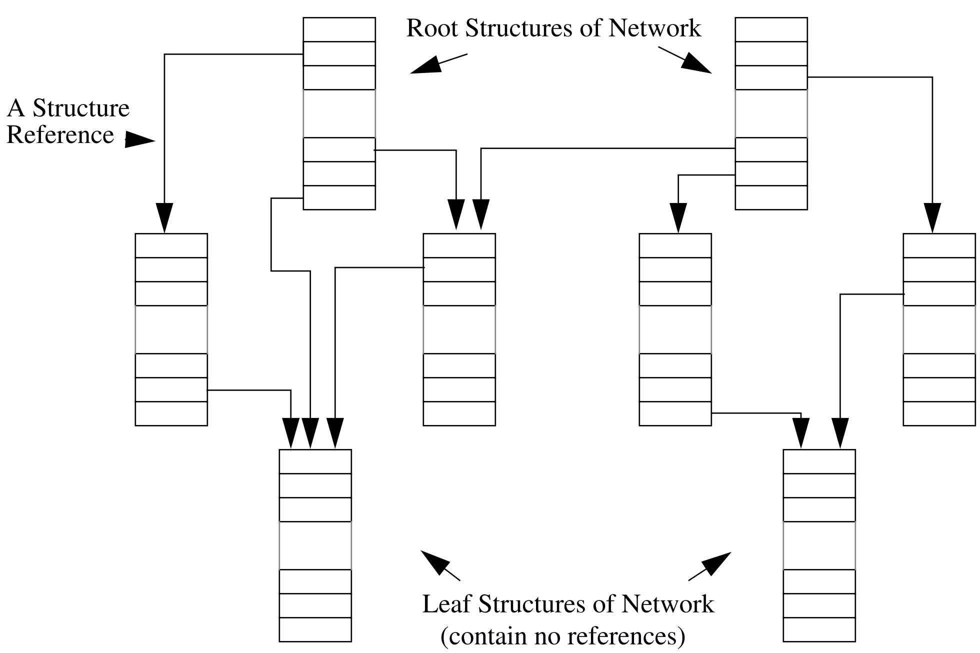
Structures

A *structure* is a resource which stores output commands for later execution. The PEX client program places output commands in a structure where they are referred to as *structure elements*. Structures can reference other structures; such a hierarchy of structures is called a *structure network*. You must be able to represent structure networks as acyclic graphs, therefore structures should not reference structures recursively, either directly or indirectly. Structures in a structure network that have no references to them are called *root structures* of the network. Structures that contain no references to other structures are called *leaf nodes* of the network. For many structure network operations, such as drawing a structure network, you select a *sub-network* of a structure network by designating any structure as the root structure for this subsequent operation. The implementation performs the operation, treating the designated structure as the root structure of a network.

Here is an example of a structure which is also a simple structure network:



Here is an example of a more complex structure network with two root structures of the network. Each structure element that references another structure must be of the type Execute Structure, Conditional Execute Structure, Include Structure, or Conditional Include Structure:



A structure is an X11 resource and carries all the responsibilities and access rights of X11 resources.

You render the structure elements within a structure in one of the following ways:

- Invoke the **PEXRenderNetwork** function
- Invoke the **PEXRenderElements** function
- Invoke one of the following OC functions while the renderer is not in **PEXIdle** state (in immediate mode): **PEXOCCExecuteStructure**, **PEXOCCConditionalExecuteStructure**, **PEXOCCIncludeStructure**, **PEXOCCConditionalIncludeStructure**.

Use **PEXGetStructureInfo** to obtain the values of the following structure attributes. You modify the first three attributes directly with functions described later in this chapter. However, you may also change the element pointer implicitly with many of the structure editing functions. You cannot change any of the last three attributes directly. Their values change as you edit the structures by inserting and/or deleting elements.

- element pointer
- editing mode
- permission
- number of structure elements
- length (in units of four bytes)
- whether or not referenced by other structures

Create a structure resource by invoking the **PEXCreateStructure** function. Add structure elements to existing structures simply by calling output command functions with the appropriate structure identifier and a request type of **PEXOCStore** or **PEXOCStoreSingle**. For more information on how PEXlib generates PEX requests for storing output commands in a structure, see "Rendering and Storing Output Commands" on page 133.

Delete structure resources by calling the **PEXDestroyStructures** function. The **PEXDestroyStructures** function deletes a list of structure resources that you specify and has the side effect of deleting all other structure elements which reference any of the deleted structures.

Other functions exist for dealing with structures. These include functions to change all references to 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. Because structures can invoke 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 a root of a structure network or sub-network, then 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 output command that references structure "B," then element position zero in "B" has an element reference path of {"A":14, "B":0}. Because 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.

14.1. Positioning of the Element and Element Pointer

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.

The *element pointer* contains an offset into the structure. When a structure is first created, the element pointer has a value of zero. Many command functions use the element pointer value to get, 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*. If the absolute offset is greater than the offset of the last element in the structure, then the element pointer is left pointing to the last element in the structure. For functions which specify a range of elements with two {*whence*, *offset*} pairs, the elements affected are those between, and including, the two computed positions regardless of which position was specified first and which was second.

Conceptually, a *null element* exists before the first element and has an offset of zero and a type of **PEXOCNil**. Use the value **PEXBeginning** with an offset of zero to set the element pointer to this position. See "PEXElementSearch - Element Search (5.0)" on page 589 for a discussion on the usage of **PEXOCNil** in searches.

A label is a special structure element that allows convenient editing of a structure. A pick identifier is similar to a label. It is returned as part of the pick path and is used to aid in identifying the selected object. Labels and pick identifiers are stored as structure elements and move with other structure elements as you insert or delete elements before them in the structure. You can move the element pointer within a structure to a specific offset, a particular label element, a particular pick identifier element, or the next location of a particular type of structure element.

14.2. Inserting Structure Elements

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 creates a structure element and inserts it 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 you want to add structure elements 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.

14.3. Replacing Structure Elements

Overwrite existing structure elements 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. After each replace operation, the element pointer is updated to point to the last newly-replaced element.

Because PEXlib groups the OC's generated by your OC function calls in unpredictable ways, you often cannot be certain that a series of OC function call invocations that replace elements in a structure are handled in one request to the PEX implementation. Because the element pointer is left at the last replaced element and a replace request begins with the current element, a replace operation divided into multiple requests may cause the element at the tail end of one request to be replaced by the first element of the next request. Therefore, to be completely safe, you should use the **PEXOCStoreSingle** request type in the OC function argument list and update the element pointer after replacing each element to move the element pointer to the next element to prepare to replace it. Another, perhaps more efficient, approach is to delete *n* elements from the structure and then invoke the *n* OC functions to insert the new elements in the place of the deleted elements. You use **PEXOCStore** in replace mode only if you are certain that a string of OC function calls will generate a single request, as in the case of replacing several small attribute or primitive output commands.

14.4. Copying Structure Elements

Some structure editing commands can copy a range of structure elements. 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.

14.5. Deleting Structure Elements

Some structure editing commands can 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.

14.6. Structure Permissions

Structure permissions provide a mechanism for performance optimization by allowing the implementation to perform various transformations on a structure that improve performance. Some structure permissions restrict the operations you are allowed to perform on a structure. Table 23, "Structure Permissions and Operations," on page 578, lists the structure requests allowed for each permission. You may perform all rendering and picking requests on structures of any permission with the following exception:

- Picking on structures with permission of **PEXStructureWriteOnly** or **PEXStructureLocked** will return valid pick paths, including the offset, for Execute Structure, Conditional Execute Structure, Include Structure, and Conditional Include Structure output commands, but not necessarily for primitives. This implies that the entire pick path returned from a pick operation is valid up to the last element in the path. The offset in the last element of the path may be invalid.

A structure's permission defaults to **PEXStructureReadWrite** when you create it. Use the **PEXSetStructurePermission** function to change the permission of the structure.

Table 23: Structure Permissions and Operations

Permission	Allowed Structure Operations
PEXStructureReadWrite	All.
PEXStructureWriteOnly	All but **PEXFetchElements**, **PEXCopyElements**(see below), **PEXCopyStructure**(see below).
PEXStructureLocked	**PEXChangeStructureReferences**, **PEXDestroyStructures**, **PEXGetAncestors**, **PEXGetDescendants**, **PEXGetStructureInfo**, **PEXGetStructuresInNetwork**.

When a structure's permission is **PEXStructureWriteOnly** you cannot read the structure's elements using the **PEXFetchElements** function or copy its elements to a structure which has a permission of **PEXStructureReadWrite**. If you try any of these operations, you may receive a structure permission error. The PEX implementation is free to modify an individual element as long as the structure network, the element's type, and the element's effect during rendering and picking is maintained. Maintaining the structure network implies that structure-referencing output commands (e.g., ExecuteStructure) retain their positions in the structure as the structure's permission is changed.

You cannot edit, read, or copy elements from a structure that has its permission set to **PEXStructureLocked**.

If you lock a structure that contains Execute Structure, Conditional Execute Structure, Include Structure, or Conditional Include Structure output commands, the operation does not modify the permissions of the referenced structures. Locked structures may include references to other structures. The PEX implementation is free to modify the structure's elements as long as the structure network, the elements' types, and the elements' effects during rendering and picking are maintained. Maintaining the structure network implies that structure-referencing output commands (e.g., ExecuteStructure) retain their positions in the structure as the structure's permission is changed. Note that these references are updated by the implementation in response to **PEXChangeStructureReferences** requests or the deletion of the referenced structures as if the structure were not locked. As an example, if locked structure A contained an ExecuteStructure OC for structure B at offset 4 and an ExecuteStructure OC for structure C at offset 6, and you destroy structure B, the ExecuteStructure OC at offset 4 is deleted and the ExecuteStructure for structure C at offset 6 moves to offset 5. For the purposes of rendering and picking, the only valid element positions in a locked structure are position 0 and the positions of structure references. When you use functions that require specific element positions, such as picking, searching, partial traversals, and structure editing on locked structures, you are restricted to using element positions that refer to position 0 or to positions that contain structure referencing output commands.

14.6.1. Changing Structure Permissions

You can change the permission of a newly created structure, or a structure whose permission has not yet been modified, to the permissions **PEXStructureWriteOnly** or **PEXStructureLocked**. If you change a structure's permission to **PEXStructureWriteOnly**, you can only change it to the permission **PEXStructureLocked**. If you change a structure's permission to **PEXStructureLocked** then you can no longer alter that structure's permission. Note that it is an error to attempt to change the permission of a structure to the structure's current permission.

Table 24: Allowed Structure Permission Changes

	New Permission	
Current Permission	**PEXStructureWriteOnly**	**PEXStructureLocked**
PEXStructureReadWrite	OK	OK
PEXStructureWriteOnly	Error	OK
PEXStructureLocked	Error	Error

PEXChangeStructureRefs - Change Structure References (5.0)

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 [§2.3.19, page 29].
new_structure	The resource identifier of the structure now referenced [§2.3.19, page 29].

Returns

None.

Description

Invoke **PEXChangeStructureRefs** to modify all structure referencing elements in all of the existing structures which reference the structure identified by *old_structure*. The references are modified to reference the structure identified by *new_structure* instead. Existing references to *new_structure* are not affected.

For PHIGS workstation resources, if *new_structure* is already posted, then its posting is not modified. On PHIGS workstation resources where *old_structure* is posted but *new_structure* is not, a posting is added for *new_structure* at the same priority as *old_structure*. *old_structure* is unposted on all PHIGS workstation resources where it is posted. Both structures must already exist as valid structure resources.

If *new_structure* does not exist, then this function issues an error and no changes are made to any references to *old_structure*, if they exist.

Errors

BadPEXStructure	A specified structure resource identifier is invalid.

See Also

PEXCreateStructure, **PEXGetAncestors**, **PEXGetDescendants**, **PEXGetStructureInfo**, **PEXGetStructuresInNetwork**

PEXCopyElements - Copy Elements (5.0)

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 [§2.3.19, page 29].
src_whence1	A value specifying, with *src_offset1*, the first limit of the range of elements to be copied (**PEXBeginning**, **PEXCurrent** or **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** or **PEXEnd**).
src_offset2	The offset from *src_whence*2 denoting the second limit of the range of elements to be copied.
dst_structure	The resource identifier of the destination structure [§2.3.19, page 29].
dst_whence	A value specifying, with *dst_offset*, the position at which the elements are inserted into the destination structure (**PEXBeginning**, **PEXCurrent** or **PEXEnd**).
dst_offset	The offset from *dst_whence* denoting the position at which the elements are inserted into the destination structure.

Returns

None.

Description

Invoke **PEXCopyElements** to copy a range of elements from *src_structure* to *dst_structure*.

If a computed offset is less than zero, then this function sets the offset to zero before obtaining the element information. If a computed offset is greater than the number of elements in the structure, then this function sets the offset to the offset of the last element in the structure. If element zero is specified as the start of the element range to be copied from *src_structure*, then the implementation ignores element zero and begins the copy operation with the first element.

The same structure can be named as *src_structure* and *dst_ structure*. 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, this function updates the element pointer of *dst_structure* to point at the last element copied into it. The editing mode attribute of *dst_structure* is ignored during this request. This function always inserts the copied elements into *dst_structure* and never uses them to replace existing structure elements.

Errors

BadPEXStructure	A specified structure resource identifier is invalid.
BadPEXStructurePermission	Elements in a structure with a permission of **PEXStructureWriteOnly** cannot be copied to a structure which has permission set to **PEXStructureReadWrite**. Structure elements also cannot be copied if the source structure or destination structure has the permission set

	to **PEXStructureLocked**.
BadValue	A specified value for a *whence* argument is invalid.

See Also

PEXCreateStructure

PEXCopyStructure - Copy Structure (5.0)

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 [§2.3.19, page 29].
dst_structure	The resource identifier of the destination structure [§2.3.19, page 29].

Returns

None.

Description

Invoke **PEXCopyStructure** to copy elements in the source structure (*src_structure*) to the destination structure (*dst_structure*).

Both structures must already exist as valid structure resources. This function deletes elements that are already in the *destination structure*. This function copies the element pointer and editing mode attributes of the *source structure* to the *destination structure* as well. This function does not modify the permission of the *destination structure*.

Errors

BadPEXStructure	A specified structure resource identifier is invalid.
BadPEXStructurePermission	Elements in a structure with a permission of **PEXStructureWriteOnly** cannot be copied to a structure which has a permission set to **PEXStructureReadWrite**. The structure also cannot be copied if the source structure or destination structure has the permission set to **PEXStructureLocked**.

See Also

PEXCreateStructure

PEXCreateStructure - Create Structure (5.0)

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 [§2.3.19, page 29].

Description

Invoke **PEXCreateStructure** to create a structure resource and get the resource identifier of the created structure. The returned identifier is used to refer to the created structure resource.

Errors

BadAlloc	The implementation failed to allocate the resource.

See Also

PEXDestroyStructures

PEXDeleteBetweenLabels - Delete Elements Between Labels (5.0)

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 [§2.3.19, page 29].
label1	The first label.
label2	The second label.

Returns

None.

Description

Invoke **PEXDeleteBetweenLabels** to delete a range of elements from the specified *structure*.

This function deletes the elements between the two labels. The *label* elements themselves are not deleted. A search for the first label (*label1*) starts at the current offset plus one. A search for the second label (*label2*) starts at the element following the first label. After the deletion operation, this function sets the structure element pointer to the pointer position at the first label (*label1*).

If this function does not find either of the two specified labels between the starting point of the search and the end of the structure, then no deletion occurs and the structure's element pointer is left unchanged.

Errors

BadPEXLabel	One or both labels were not found.
BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The structure's permission is **PEXStructureLocked**.

See Also

PEXCreateStructure, **PEXOCCLabel**

PEXDeleteElements - Delete Elements (5.0)

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 [§2.3.19, page 29].
whence1	A value specifying, with *offset1*, the first limit of the range of elements to be deleted (**PEXBeginning**, **PEXCurrent** or **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** or **PEXEnd**).
offset2	The offset from *whence2* denoting the second limit of the range of elements to be deleted.

Returns

None.

Description

Invoke **PEXDeleteElements** to delete a range of elements from the structure specified by *structure*.

If a computed offset is less than zero, then this function sets the offset to zero before obtaining the element information. If a computed offset is greater than the number of elements in the structure, then this function sets the offset to the offset of the last element in the structure. If you delete a null element set, then the result is effectively a no-op. Position 0 may be in the range, but no element is deleted for that position.

After the deletion operation, this function sets the structure element pointer to the element immediately preceding the range of deleted elements.

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The structure's permission is **PEXStructureLocked**.
BadValue	The specified value for a *whence* argument is invalid.

See Also

PEXCreateStructure

PEXDeleteToLabel - Delete Elements to Label (5.0)

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 [§2.3.19, page 29].
whence	A value specifying, with *offset*, the beginning of the range of elements to be deleted (**PEXBeginning**, **PEXCurrent** or **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

Invoke **PEXDeleteToLabel** to delete 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, then no elements are deleted.

If the computed offset is less than zero, then it is set to zero before the deletion occurs. If the computed offset is greater than the number of elements in the structure, then the offset is set to the offset of the last element. After a successful deletion, the structure element pointer is set to the computed offset specified by *whence* and *offset*.

Errors

BadPEXLabel	The specified label does not exist.
BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The structure's permission is **PEXStructureLocked**.
BadValue	The specified value for *whence* argument is invalid.

See Also

PEXCreateStructure, **PEXOCCLabel**

PEXDestroyStructures - Destroy Structures (5.0)

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 [§2.3.19, page 29].

Returns

None.

Description

Invoke **PEXDestroyStructures** to delete each structure in the list of structure identifiers, remove all references to each from the other structures maintained by the implementation, and free all memory associated with each structure in the list.

This function also removes any structure elements of type Execute Structure, Include Structure, Conditional Execute Structure, and Conditional Include Structure 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 function call, then its element pointer continues to point at the same structure element. However, if the structure element being pointed at is removed, then the element pointer is positioned at the previous structure element.

You still can inquire any paths in search contexts or pick measures that contain references to a deleted structure. 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, then the implementation issues a a path error.

Errors

BadPEXStructure	A specified structure resource identifier is invalid.

See Also

PEXChangeStructureRefs, PEXCreateStructure, PEXGetAncestors, PEXGetDescendants, PEXGetStructuresInNetwork

PEXElementSearch - Element Search (5.0)

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 [§2.3.19, page 29].
whence	A value specifying, with *offset*, the offset at which the search is to begin (**PEXBeginning**, **PEXCurrent** or **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 element types in the inclusion array.
incl_list	An array of element types to search for.
excl_count	The number of element types in the exclusion array.
excl_list	An array of element types not to search 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 is either **PEXFound** or **PEXNotFound** depending upon the result of the search.

Description

Invoke **PEXElementSearch** to search for the first occurrence of the specified element type in the specified *structure*. The search always includes the starting element.

If the computed starting offset is less than zero, then it is set to zero before the search is performed. If the computed starting offset is greater than the number of elements in the structure, then it is set to the offset of the last element in the structure.

An element is selected if its element type is contained in the inclusion list (*incl_list*) and is not contained in the exclusion list (*excl_list*). An element type of **PEXOCAll** selects all element types, including **OCNil**. **OCNil** refers to the type of the implicit element at position 0. If a structure element type is in both the *inclusion list* 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 referenced structures. If the search finds a match, then the function returns a return status of **PEXFound** and the offset of the matching element. If the search is unsuccessful, then the function returns a return status of **PEXNotFound**.

The element pointer position of structure is not changed, regardless of the returned search status.

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The structure's permission is **PEXStructureLocked**.
BadValue	The specified value for a *whence* or *direction* argument is invalid.

See Also

PEXCreateStructure

PEXFetchElements - Fetch Elements (5.0)

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 [§2.3.19, page 29].
whence1	A value specifying, with *offset1*, the first limit of the range of elements to be fetched (**PEXBeginning**, **PEXCurrent** or **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** or **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** or **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

Invoke **PEXFetchElements** to get a range of structure elements from the specified *structure*.

If either computed offset is less than zero, then it is set to zero before getting the structure elements. If either computed offset is greater than the number of elements in the structure, then it is set to the offset of the last element in the structure. The element pointer attribute of *structure* is not affected by this command. No information is returned for inquiries on element offset zero.

A null pointer is returned if the implementation does not support the requested floating-point format.

Any text or annotation text output commands returned are formatted as encoded text or encoded annotation text.

PEXlib allocates memory for the return value. Invoke **XFree** to deallocate the memory.

Errors

BadPEXFloatingPointFormat	The specified floating-point format is invalid or unsupported.
BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The structure's permission is **PEXStructureLocked** or **PEXStructureWriteOnly**
BadValue	The specified value for a *whence* argument 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 [§2.3.19, page 29].
whence1	A value specifying, with *offset1*, the first limit of the range of elements to be fetched (**PEXBeginning**, **PEXCurrent** or **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** or **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

Invoke **PEXFetchElementsAndSend** to get a range of structure elements from the specified *structure*. This function is like **PEXFetchElements** except this function does not return the list of output commands to the application but sends them directly to *dst_display*.

Invoking this function is similar to invoking **PEXFetchElements**, and then sending the returned list of output commands by invoking **PEXStartOC**, **PEXCopyBytesToOC** and **PEXFinishOC**.

If either computed offset is less than zero, then it is set to zero before getting the structure elements. If either computed offset is greater than the number of elements in the structure, then it is set to the offset of the last element in the structure. The element pointer attribute of *structure* is not affected by this command. No information is returned for inquiries on element offset zero.

If the *dst_display* does not support the same floating-point format as the format PEXlib is using with the *src_display*, and if PEXlib can not convert to a format supported by the *dst_display*, then the function returns unsuccessfully.

Sending output commands to a structure which has an editing mode of **PEXStructureReplace** is not really useful. The behavior is unpredictable unless you use a request type of **PEXOCStoreSingle**. And, if the request type is **PEXOCStoreSingle**, each output command simply replaces 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, then the application can delete those elements first, and then insert the new ones.

Errors

BadPEXStructure	A specified structure resource identifier is invalid.
BadPEXStructurePermission	The source structure's permission is **PEXStructureLocked** or **PEXStructureWriteOnly**, or the destination structure is **PEXStructureLocked**.
BadValue	The specified value for a *whence* or *req_type* argument 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 [§2.28.4, page 64].

Returns

None.

Description

Invoke **PEXFreeStructurePaths** to deallocate memory returned by **PEXGetAncestors**, **PEXGetDescendants**, and **PEXSearchNetwork**.

Errors

None.

See Also

PEXGetAncestors, **PEXGetDescendants**, **PEXSearchNetwork**

PEXGetAncestors - Get Ancestors (5.0)

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 [§2.3.19, page 29].
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 unique paths found.

Returns

An array of structure paths defining the ancestors of the specified structure [§2.28.4, page 64]; a null pointer if unsuccessful.

Description

Invoke **PEXGetAncestors** to get 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. Full paths start with the root structures and work down to a final entry with a structure identifier equal to *structure* and an offset of zero. Only unique paths are returned; in other words there are no duplicates in the list of returned paths.

PEXlib defines the following values for *path_part*:

PEXTopPart - requests that the top of the structure paths be returned.

PEXBottomPart - 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 you set the *path depth* to 1, then only one element reference is returned for each path. A *path depth* of 2 returns two elements, and so on. If you set the *path depth* to 0, then the entire path is returned.

If you specify a *path depth* of 0, then this function returns all unique paths from the root structures of the network to *structure*. If you specify a *path depth* of 1 and a *path part* of **PEXTopPart**, then this function returns the root structures of all structure networks which contain *structure*. Specifying 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. Invoke **PEXFreeStructurePaths** to deallocate the memory.

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The specified value for *path_part* is invalid.

See Also

PEXCreateStructure, **PEXGetDescendants**, **PEXGetStructuresInNetwork**

PEXGetDescendants - Get Descendants (5.0)

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 [§2.3.19, page 29].
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 unique paths found.

Returns

An array of structure paths defining the descendants of the specified structure [§2.28.4, page 64]; a null pointer if unsuccessful.

Description

Invoke **PEXGetDescendants** to get 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. Full paths start with the given structure and the offset of an element referencing another structure and work down to a final entry with the structure identifier of a leaf node and an offset of zero. Only unique paths are returned; in other words, there are no duplicates in the returned paths.

PEXlib defines the following values for *path_part*:

PEXTopPart - requests that the top of the structure paths be returned.

PEXBottomPart - 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 you set the *path depth* to 1, then only one element reference is returned for each path. A *path depth* of 2 returns two elements, and so on. If you set the *path depth* to 0, then the entire path is returned.

For instance, if you specify a *path depth* of 0**,** then this function returns all unique paths from *structure* to leaf nodes in the structure network tree. If you specify a *path depth* of 1 and a *path part* of **PEXBottomPart**, then this function determines the bottom-most structures of the structure network rooted at *structure*.

PEXlib allocates memory for the returned descendant information. Invoke **PEXFreeStructurePaths** to deallocate the memory.

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The specified value for *path_part* is invalid.

See Also

PEXCreateStructure, PEXGetAncestors, PEXGetStructuresInNetwork

PEXGetElementInfo - Get Element Information (5.0)

Note: You should use **PEXGetLargeElementInfo** instead of this function in order to handle large output commands properly.

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 [§2.3.19, page 29].
whence1	A value specifying, with *offset1*, the first limit of the range of queried elements (**PEXBeginning**, **PEXCurrent** or **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** or **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 (**PEXIEEE_754_32**, **PEXDEC_F_Floating**, **PEXIEEE_754_64** or **PEXDEC_D_Floating**).
count_return	Returns the number of element info records returned.
info_return	Returns an array of element info records describing the elements in the specified range [§2.28.1, page 63].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetElementInfo** to get information about a range of elements from the specified *structure*.

If a computed offset is less than zero, then it is set to zero before obtaining the element information. If a computed offset is greater than the number of elements in the structure, then it is set to the offset of the last element in the structure. The element pointer attribute of *structure* is not affected by this command. No information is returned for inquiries on element offset zero.

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. The size returned for each element is undefined if the structure's permission is **PEXStructureWriteOnly**.

This function returns a value of zero in the *length* field if the OC is greater than 64K words in length. To work with these large OC's, invoke the **PEXGetLargeElementInfo** function to obtain their actual lengths.

PEXlib allocates memory for the return value. Invoke **XFree** to deallocate the memory.

Errors

BadPEXFloatingPointFormat	The specified floating-point format is invalid or unsupported.
BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The structure's permission is **PEXStructureLocked**.

BadValue	The specified value for a *whence* argument is invalid.

See Also

PEXCreateStructure, PEXGetLargeElementInfo

PEXGetLargeElementInfo - Get Element Information (5.2)

Synopsis

Status **PEXGetLargeElementInfo**(Display **display*, `PEXStructure` *structure*, int *whence1*, long *offset1*, int *whence2*, long *offset2*, int *float_format*, unsigned long **count_return*, `PEXLargeElementInfo` ***info_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure [§2.3.19, page 29].
whence1	A value specifying, with *offset1*, the first limit of the range of queried elements (**PEXBeginning**, **PEXCurrent** or **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** or **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 (**PEXIEEE_754_32**, **PEXDEC_F_Floating**, **PEXIEEE_754_64** or **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 [§2.28.1, page 63].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetLargeElementInfo** to get information about a range of elements from the specified *structure*. Use **PEXGetLargeElementInfo** instead of **PEXGetElementInfo** in PEXlib 5.2 applications to handle large output commands properly.

If a computed offset is less than zero, then it is set to zero before obtaining the element information. If a computed offset is greater than the number of elements in the structure, then it is set to the offset of the last element in the structure. The element pointer attribute of *structure* is not affected by this command. No information is returned for inquiries on element offset zero.

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. The size returned for each element is undefined if the structure's permission is **PEXStructureWriteOnly**.

PEXlib allocates memory for the return value. Invoke **XFree** to deallocate the memory.

Errors

BadPEXFloatingPointFormat	The specified floating-point format is invalid or unsupported.
BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The structure's permission is **PEXStructureLocked**.
BadValue	The specified value for a *whence* argument is invalid.

See Also

PEXCreateStructure, **PEXGetElementInfo**

PEXGetStructureInfo - Get Structure Information (5.0)

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 [§2.3.19, page 29].
float_format	The floating-point format to use when computing element sizes (**PEXIEEE_754_32**, **PEXDEC_F_Floating**, **PEXIEEE_754_64** or **PEXDEC_D_Floating**).
value_mask	A mask indicating which values to return.
info_return	Returns information about the structure resource [§2.28.3, page 63].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetStructureInfo** to get information about the specified structure resource.

The *value_mask* is constructed by or’ing together the following constants:

PEXElementPtr
PEXNumElements
PEXLengthStructure
PEXHasRefs
PEXEditMode
PEXPermission (5.2)

The length of the structure is given in the number of 4-byte units, and is based upon the specified floating-point format.

If the structure’s permission is **PEXStructureWriteOnly**, then the *size* member of the PEXStructureInfo structure is an undefined value. If the structure’s permission is **PEXStructureLocked**, then the members of the PEXStructureInfo data structure *element_count, size, edit_mode,* and *element_pointer*, have undefined values.

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 (5.0)

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 [§2.3.19, page 29].
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 [§2.3.19, page 29]; a null pointer if unsuccessful.

Description

Invoke **PEXGetStructuresInNetwork** to get a list of unique structure resource identifiers that are referenced in the structure network rooted at the specified root *structure*.

PEXlib defines the following values for *which*:

> **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 is always returned in the list, unless it is an invalid structure identifier, in which case the returned list is empty.

PEXlib allocates memory for the return value. Invoke **XFree** to deallocate the memory.

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The specified value for *which* is invalid.

See Also

PEXCreateStructure, **PEXGetAncestors**, **PEXGetDescendants**

PEXSetEditingMode - Set Structure Editing Mode (5.0)

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 [§2.3.19, page 29].
mode	The editing mode (**PEXStructureInsert** or **PEXStructureReplace**).

Returns

None.

Description

Invoke **PEXSetEditingMode** to set the editing mode for the specified *structure*.

The editing mode specifies how editing operations affect the structure. PEXlib defines the following editing modes (*mode*):

> **PEXStructureInsert** - subsequent requests to create structure elements cause elements to be inserted into the structure. This is the mode when a structure is created.
>
> **PEXStructureReplace** - output requests that create structure elements cause structure elements to replace elements, starting at the location specified by the element pointer. See the discussion in "Replacing Structure Elements" on page 577 for some important considerations when replacing a series of elements.

The element pointer is then incremented by the number of elements inserted or replaced.

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The structure's permission is **PEXStructureLocked**.
BadValue	The specified value for *mode* is invalid.

See Also

PEXCreateStructure

PEXSetElementPtr - Set Structure Element Pointer (5.0)

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 [§2.3.19, page 29].
whence	A value specifying with *offset*, the element pointer position (**PEXBeginning**, **PEXCurrent** or **PEXEnd**).
offset	The offset from *whence* which specifies the element pointer position.

Returns

None.

Description

Invoke **PEXSetElementPtr** to set the element pointer for the specified *structure* to the specified position.

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

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The structure's permission is **PEXStructureLocked**.
BadValue	The specified value for *whence* argument is invalid.

See Also

PEXCreateStructure

PEXSetElementPtrAtLabel - Set Structure Element Pointer at Label (5.0)

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 [§2.3.19, page 29].
label	The value of the label.
offset	The offset from the label.

Returns

None.

Description

Invoke **PEXSetElementPtrAtLabel** to set the element pointer for the specified *structure* at a position denoted by the *label*.

A search is conducted for the next label element with the specified *label*, starting at the current element pointer position plus one and proceeding in the forward direction. The search is non-descending, that is, it does not include any structures referenced by *structure*. If *label* is found, then the element pointer for the structure is set to the location of the *label* plus *offset*, which may be negative, zero, or positive. If the computed position is less than zero, then the element pointer is set to 0. If the computed position is beyond that of the last element in the structure, then the element pointer is set to the position of the last element in the structure. If *label* is not found, then the structure's element pointer is left unchanged.

Errors

BadPEXLabel	The specified label does not exist.
BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The structure's permission is **PEXStructureLocked**.

See Also

PEXCreateStructure, **PEXOCCLabel**

PEXSetElementPtrAtPickID - Set Structure Element Pointer at Pick ID (5.2)

Synopsis

void **PEXSetElementPtrAtPickID**(Display **display*, `PEXStructure` *structure*, long *pick_id*, long *offset*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure [§2.3.19, page 29].
pick_id	The value of the pick identifier.
offset	The offset from the pick identifier.

Returns

None.

Description

Invoke **PEXSetElementPtrAtPickID** to set the element pointer for the specified *structure* at a position denoted by the *pick_id*.

A search is conducted for the next occurrence of a pick identifier structure element with the specified *pick_id*, starting at the current element pointer position plus one and proceeding in the forward direction. The search is non-descending, that is, it does not include any structures referenced by *structure*. If the *pick_id* element is found, then the element pointer for the structure is set to the location of the *pick_id* element plus *offset*, which may be negative, zero, or positive. If the computed position is less than zero, then the element pointer is set to 0. If the computed position is beyond that of the last element in the structure, then the element pointer is set to the position of the last element in the structure. If the *pick_id* element is not found, then the implementation issues a *BadPEXPickID* error and the function leaves the structure's element pointer unchanged.

Errors

BadPEXPickID	The specified pick ID does not exist.
BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The structure's permission is **PEXStructureLocked**.

See Also

PEXCreateStructure, **PEXOCCPickID**

PEXSetStructurePermission - Set Structure Permission (5.2)

Synopsis

void **PEXSetStructurePermission**(Display **display*, `PEXStructure` *structure*, int *permission*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure [§2.3.19, page 29].
permission	The structure permission (**PEXStructureWriteOnly** or **PEXStructureLocked**).

Returns

None.

Description

Invoke **PEXSetStructurePermission** to set the permission of the specified *structure*.

PEXlib defines the following values for structure *permission*:

PEXStructureWriteOnly - indicates you can write elements to the structure (but you cannot copy its elements to a structure which has a permission of **PEXStructureLocked**.

PEXStructureLocked - indicates you cannot edit, inquire, or copy elements from the structure.

For more information, see "Structure Permissions" on page 578.

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXStructurePermission	The specified structure *permission* is invalid. See "Allowed Structure Permission Changes" on page 579.
BadValue	The value in *permission* is invalid.

See Also

Chapter 15

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. You cannot directly modify the contents of the current name set nor query the contents 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 the 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, then the name 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, then the scene is drawn as normal. If the user invokes **PEXChangeNameSet** to add “antique” to the highlighting inclusion set, then 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, then all “wooden” objects and all “antique” objects are highlighted. If “wooden” is added to the highlighting exclusion set instead, then 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 implementation may limit the set of available names. To determine the number of names available, invoke the **PEXGetImpDepConstants** function. PEXlib includes functions to create, copy, free, query, and modify name sets.

PEXChangeNameSet - Change Name Set (5.0)

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 [§2.3.13, page 28].
action	Type of change to be made (**PEXNSAdd**, **PEXNSRemove** or **PEXNSReplace**).
count	The number of names.
names	An array of names [§2.3.12, page 28].

Returns

None.

Description

Invoke **PEXChangeNameSet** to modify the specified *name set* resource.

PEXlib supports the following values for *action*:

PEXNSAdd - adds the list of names to the name set.

PEXNSRemove - removes the specified name from the name set.

PEXNSReplace - replaces the names currently in the name set.

If you request to remove a name that does not exist, then this function is silently ignored.

Errors

BadPEXNameSet	The specified name set resource identifier is invalid.
BadValue	The specified value for *action* argument is invalid.

See Also

PEXCreateNameSet, **PEXGetNameSet**

PEXCopyNameSet - Copy Name Set (5.0)

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 [§2.3.13, page 28].
dst_nameset	The resource identifier of the destination name set [§2.3.13, page 28].

Returns

None.

Description

Invoke **PEXCopyNameSet** to copy the contents of the source name set (*src_nameset*) to the destination name set (*dst_nameset*). Both must already exist as valid name set resources.

This function overwrites all values already in the *destination name set*.

Errors

BadPEXNameSet	The specified name set resource identifier is invalid.

See Also

PEXCreateNameSet

PEXCreateNameSet - Create Name Set (5.0)

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 [§2.3.13, page 28].

Description

Invoke **PEXCreateNameSet** to create a name set resource and get the resource identifier of the created name set.

Use the returned identifier to refer to the created name set.

Errors

BadAlloc	The implementation failed to allocate the resource.

See Also

PEXFreeNameSet

PEXFreeNameSet - Free Name Set (5.0)

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 [§2.3.13, page 28].

Returns

None.

Description

Invoke **PEXFreeNameSet** to delete 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 (5.0)

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 [§2.3.13, page 28].
count_return	Returns the number of names.
names_return	Returns an array of names [§2.3.12, page 28].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetNameSet** to get the names in the specified name set.

PEXlib allocates memory for the return value. Invoke **XFree** to deallocate the memory.

Errors

BadPEXNameSet	The specified name set resource identifier is invalid.

See Also

PEXChangeNameSet, **PEXCreateNameSet**

Chapter 16

Texture Mapping

Texture mapping simulates material properties for surface primitives by adding visual detail with color. This is accomplished by blending the sampling of a map of colors with the primitive's color.

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

Texture maps are defined by creating texture map PEX resources. Only color mipmap texture maps are supported. A color mipmap is a pre-computed area sampling mechanism which fixes the cost of approximating the average color over a large number of texels (the individual components of a texture map). It is created as a pyramid of down-sampled levels, from the "base" to the typically single-texel "pinnacle". Each level is a 1D, 2D, or 3D array of texels, according to the dimension of the texture map. Typically each texel in level *n* equals the average of four texels beneath it in level *n*-1. Thus the resolution of each map typically becomes half of the preceding level until the top level, or *pinnacle*, is reached, which typically has one texel representing the average color of the original base level map. So, if multiple levels are defined, then an array of texels is defined for each level.

The texel types are luminance, luminance and alpha, red-green-blue, and red-green-blue-alpha, in a variety of data formats. A texture map is created with function **PEXCreateMipmapTM**, which accepts the color mipmap levels as arrays of texels. Alternately, the function **PEXCreateMipmapTMFromResources** creates a color mipmap from a list of resources, such as drawables, image buffers, pixmaps, or photomaps, where each resource defines one level of the mipmap. A second list of resources is used if the texels include an alpha component.

A texture map resource is enabled by referencing it from an entry of the texture mapping lookup table and enabling that entry with the pipeline state's *active textures*, modified with the **PEXOCCSetSurfaceActiveTextures** OC. The corresponding *bf active textures* pipeline state field, modified with the **PEXOCCSetBFSurfaceActiveTextures** OC, is applied to back-facing surfaces when face distinguishing is enabled. An application can select multiple texture maps.

16.1. The Texture Mapping Lookup Table

There are many controls which are applied to a texture map. Texture maps are referenced through a lookup table to allow multiple combinations of control values to be easily applied to the same texture map. The table mechanism also facilitates ensuring control values, many of which are dependent on the specific texture map, are valid. See "Texture Mapping Table" on page 460.

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

Several of the fields control parameterization, that is, how texels are mapped to a facet. The *parameterization method* field specifies how the texture map coordinates for each vertex are determined. The texture map coordinates select a position in the texture map. The portion of a texture map selected by the vertices's texture map coordinates is then applied to the facet. The **PEXTMParamVertexCoordMC**, **PEXTMParamVertexCoordWC**, **PEXTMParamVertexNormalMC**, **PEXTMParamVertexNormalWC**, **PEXTMParamNURBUV**, and **PEXTMParamFloatData** parameterization methods use the vertex, normal, u-v parameterization, or floating-point data list data directly as the texture map coordinates. Field *fp data index* of the texture mapping lookup table defines the index within the floating-point data list of the first of the consecutive texture mapping coordinates when the parameterization method is **PEXTMParamFloatData**. The **PEXTMParamReflectionSphereVRC** and **PEXTMParamReflectionSphereWC** parameterization methods select texture map coordinates by picking them off a conceptual sphere surrounding the facet. A special reflection vector calculated for each vertex is used to select a point on the sphere. The resulting texture mapping is often called environment mapping since the texturing is view dependent

- the facet appears to reflect fixed objects as the facet is moved around. The *reflection matrix and hints* field of the texture mapping lookup table is applied only when one of these environment mapping parameterization methods is selected. The reflection matrix is used to transform the reflection vectors before they select the texture coordinates from the sphere, which has a fixed orientation.

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

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

Textures sometimes map to something less than the entire facet. The *t0, t1,* and *t2 boundary condition* fields specify how the region in each direction outside the mapped texture is handled. PEXlib defines the following boundary conditions:

PEXTMBoundaryWrap - indicates the texture is repeated across the facet.

PEXTMBoundaryMirror also indicates the texture is repeated, but alternately in reverse order.

PEXTMBoundaryClampBorder - indicates the closest border texel is used to determine the sampled color value beyond that border.

PEXTMBoundaryClampExplicit - indicates the *boundary clamp method* field is used to determine the boundary condition. (This level of indirection exists to prevent ambiguity in regions which extend outside the mapped texture in more than one direction.)

PEXlib defines the following clamping methods:

PEXTMBoundaryClampExplicitDiscontinue - boundary clamp method that indicates texturing is to be discontinued beyond the mapped texture, that is, the primitive's color before applying that texture map is preserved.

PEXTMBoundaryClampExplicitColor - boundary clamp method that indicates the *boundary clamp color* field specifies the texture's color value.

For the sampling methods that use multiple texels on a mipmap level to compute a weighted average, it is possible that the neighboring texels do not exist because the implementation is sampling near the texture map boundary. In these cases, the implementation synthesizes the missing texels by using the boundary condition in effect. However, a boundary condition of **PEXTMBoundaryClampExplicit** with a clamping method of **PEXTMBoundaryClampExplicitDiscontinue** does not cause the primitive's intrinsic color to be included in the weighted average.

The remaining fields in the texture mapping lookup table specify the blending of the color values derived from the texture map and boundary conditions with the primitive's color. The *composition method* field specifies how this blending is done:

PEXTMCompReplace - indicates the primitive's color is replaced with the derived texture color. The alpha component is set to 1.0 if the texture map didn't include an alpha component.

PEXTMCompModulate - specifies the primitive's color and derived texture color are multiplied.

PEXTMCompBlendEnvColor - indicates the derived texture color is multiplied by the color specified in the *environment color* field. The result is added to the multiplication of the primitive's color and the inverse of the derived texture color. The primitive's alpha component is just multiplied by the derived texture alpha component (1.0 if not present).

PEXTMCompDecal - specifies the derived texture color is multiplied by its alpha component. The result is added to the multiplication of the primitive's color and the inverse of the derived texture alpha component. The primitive's alpha component is not modified.

PEXTMCompDecalBackground is identical to method **PEXTMCompDecal** except the color specified in the *background color* field is used instead of the primitive's color.

16.2. Texture Mapping Attribute OCs

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

The pipeline state's *tm perspective correction* field, modified by the **PEXOCCTMPerspectiveCorrection** OC, determines if, and how, texture coordinate values are adjusted when the view is a perspective projection. The choices are to correct at each vertex, to correct at each pixel, or not to correct.

The pipeline state's *tm sample frequency* field, modified by the **PEXOCCTMSamplingFrequency** OC, specifies the frequency at which the texture map is sampled. This can be set to sample at each pixel (**PEXTMSampleFrequencyPixel**), or to base the sample frequency on the surface interpolation method (**PEXTMSampleFrequencyInterpDep**), ranging between sampling once per facet for interpolation method **PEXSurfaceInterpNone**, to once per vertex of method **PEXSurfaceInterpColor**, to once per pixel for methods **PEXSurfaceInterpDotProduct** and **PEXSurfaceInterpNormal**.

The *tm resource hints* field of the pipeline state, modified by the **PEXOCCTMResourceHints** OC, enables the client to suggest how the server could most efficiently manage the texture map resources. The client expresses a preference for whether speed (**PEXTMResourceHintsSpeed**) or space (**PEXTMResourceHintsSpace**) is more critical, or whether it doesn't matter (**PEXTMResourceHintsNone**). The client also can provide a prioritized usage list of texture mapping table entries.

16.3. Texture Map Resource Management Functions

There are two functions that enable you to create texture maps using data that you provide. If the data is on the client-side, then use the **PEXCreateMipmapTM** function to send the data to the PEX implementation to create a texture for later use.

In other cases, the data that you want to use to create the texture is stored in the X server as the contents of an X resource such as pixmaps or windows. Here it is efficient to invoke the **PEXCreateMipmapTMFromResources** function.

PEXCreateMipmapTM - Create Mipmap Color Texture Map (5.2)

Synopsis

`PEXTextureMap` **PEXCreateMipmapTM**(Display **display*, int *dimensionality*, int *texel_type*, int *num_levels*, `PEXMipmapLevel` **levels*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
dimensionality	The dimension of the texture map (**PEXTMDimension1D, PEXTMDimension2D** or **PEXTMDimension3D**).
texel_type	The type of texel data in the mipmap (**PEXTexelTypeLum, PEXTexelTypeLum8, PEXTexelTypeLum16, PEXTexelTypeLumA, PEXTexelTypeLumA8, PEXTexelTypeLumA16, PEXTexelTypeRGB, PEXTexelTypeRGB8, PEXTexelTypeRGB16, PEXTexelTypeRGBA, PEXTexelTypeRGBA8** or **PEXTexelTypeRGBA16**).
num_levels	The number of levels in the mipmap.
levels	A pointer a list of `PEXMipmapLevel` structures each representing a level of the color mipmap [§2.29.14, page 66].

Returns

The resource ID for a PEX Texture Map [§2.3.22, page 29].

Description

Invoke **PEXCreateMipmapTM** to create a mipmap texture resource that you can reference from a texture mapping LUT (page 460).

The *dimensionality* argument indicates the dimension of the texture map being created. *Levels* provides the actual texel data. *Texel_type* indicates the type of texel data as well as the components of the newly created texture resource.

The *num_levels* of mipmap levels in the *levels* argument are ordered to define the texture map, starting with the base level. Each member includes the dimension in each direction of that level of the texture map. The texels are assumed to be stored in *t*0-*t*1-*t*2 order, with *t*0 varying fastest and *t*2 varying slowest. The texture coordinates are normalized, so the first through last texel correspond to the texel coordinate range 0.0 to 1.0.

Inquire *MaxFastTMSize* with the function **PEXGetImpDepConstants** to determine whether there is a maximum size in any dimension on the base level which must be respected to achieve optimized implementation. If PEX has such a limit, then it treats larger texture maps in an implementation-dependent way. PEX implementations may have other restrictions on the dimensions of each level of the texture map. Inquire *PowerOfTwoTMRequired* with **PEXGetImpDepConstants** to determine if each dimension must be a power of two. Inquire *SquareTMRequired* with the same request to determine if all the dimensions on each level must be equal to one another.

Errors

BadAlloc	The implementation failed to allocate a texture map resource.
BadValue	The specified texture map dimension is not valid.
BadValue	The specified texture map texel type is not valid.
BadValue	The specified number of texel arrays does not match the number of texel arrays expected.
BadValue	The texture map size is required to be a power of two.

BadValue	The texture map size is required to be square.
BadValue	Texel component is out of range.

See Also

PEXCreateMipmapTMFromResources, **PEXFreeTextureMap**

PEXCreateMipmapTMFromResources - Create Mipmap Color Texture Map From Resources (5.2)

Synopsis

`PEXTextureMap` **PEXCreateMipmapTMFromResources**(Display **display*, int *dimensionality*, int *texel_type*, Drawable *drawable*, unsigned short *luminance_channel*, unsigned short *alpha_channel*, int *num_levels*, `PEXTexelDimension` **dimensions*, XID **color_data*, XID **alpha_data*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
dimensionality	The dimension of the texture map (**PEXTMDimension1D, PEXTMDimension2D** or **PEXTMDimension3D**).
texel_type	The type of texel data in the mipmap (**PEXTexelTypeLum, PEXTexelTypeLum8, PEXTexelTypeLum16, PEXTexelTypeLumA, PEXTexelTypeLumA8, PEXTexelTypeLumA16, PEXTexelTypeRGB, PEXTexelTypeRGB8, PEXTexelTypeRGB16, PEXTexelTypeRGBA, PEXTexelTypeRGBA8** or **PEXTexelTypeRGBA16**).
drawable	An example drawable indicating the visual type of the resources.
luminance_channel	Indicates source of data used to form luminance data (**PEXTMChannelRed, PEXTMChannelGreen, PEXTMChannelBlue** or **PEXTMChannelNTSC**).
alpha_channel	Indicates source of data used to form alpha data (**PEXTMChannelRed, PEXTMChannelGreen, PEXTMChannelBlue** or **PEXTMChannelNTSC**).
num_levels	The number of levels in the mipmap.
dimensions	Specifies the dimensions in each direction of each level of the mipmap [§2.29.13, page 65].
color_data	List of IDs of the resources data is taken from to form texture map.
alpha_data	List of IDs of the resources data from which data is taken to form alpha component of texture map, if alpha is part of the indicated texel type.

Returns

The resource ID for a PEX Texture Map [§2.3.22, page 29].

Description

Invoke **PEXCreateMipmapTMFromResources** to create a mipmap texture resource that you can reference from a texture mapping LUT (page 460).

The implementation normalizes texture coordinates so that, in each dimension, the first through the last texel correspond to the texture coordinate range 0.0 to 1.0.

If the texture map is one-dimensional, then one color resource and, optionally, one alpha resource define each level of the texture map, starting with the base level. Texture coordinate $t0 = 0.0$ corresponds to the lower-left of the image defined by the resource. Subsequent texel values are taken from the bottom row of the resource along the x direction. The rows of the resource are conceptually wrapped, so it is possible for a texture map level's dimension to exceed the x dimension of the resource. The wrapping occurs by moving up to the left end of the next higher row as each row is exhausted.The dimension of each level of the texture map is defined by argument *dimension*, which is an array with a member for each level of the texture map. The texture map coordinates are normalized, so the first through $c0^{th}$ texel, where $c0$ is the $t0$ dimension, correspond to the $t0$ coordinate range from 0.0 to 1.0.

In the two-dimensional texture map case, again one color resource, and optionally one alpha resource, defines each level of the texture map, starting with the base level. Texture map coordinate $t0 = t1 = 0.0$ corresponds to the lower-left of the image defined by the source. That is, the bottom (last) row of the resource supplies texels for the bottom (first) row of the texture. The *x* direction of the resource corresponds to the *t*0 direction of the texture map. The texture map dimensions for a given level cannot exceed the corresponding resource dimensions.

In the three-dimension texture map case, *c2* color resources, and optionally *c2* alpha resources, define each level of the texture map. *c2,* defined by argument *dimension* for each level, is the third dimension of the level. The *c2* resources defining each level are ordered from $t2 = 0.0$ to $t2 = 1.0$. All *c2* resources defining the base level are given first, followed by all the resources defining the next level, and so on.

The *dimensionality* argument indicates the dimension of the texture map. The list of color and alpha resource IDs provides the actual texel data. The visual of the drawable specified by *drawable* indicates the source texel format.

Texel_type indicates the texel components of the newly created texture resource. If the *texel_type* indicates luminance is included, then *luminance_channel* indicates whether one of three channels of RGB are given as luminance data, or if all three channels are combined into a single luminance value. If the *texel_type* indicates that the luminance is not included (in the destination texels), then the implementation ignores the *luminance_channel*. If the *texel_type* indicates alpha is included, then the *alpha_channel* argument indicates whether one of three channels of RGB are given as alpha data, or if all three channels are combined into a single alpha value. If the *texel_type* indicates that alpha is not included, then the implementation ignores *alpha_data* and *alpha_channel*.

PEXlib defines the following values for the channel selection constants (*luminance_channel* and *alpha_channel*).

> **PEXTMChannelRed** - selects the red channel. (5.2)
>
> **PEXTMChannelGreen** - selects the green channel. (5.2)
>
> **PEXTMChannelBlue** - selects the blue channel. (5.2)
>
> **PEXTMChannelNTSC** - indicates the three channels of red, green and blue are to be combined into a single luminance or alpha value using the NTSC encoded values as follows:
> 0.299*red + 0.587*green + 0.114*blue (5.2)

Inquire *MaxFastTMSize* with the function **PEXGetImpDepConstants** to determine whether there is a maximum size for any dimension on the base level which must be respected to achieve optimized implementation. If PEX has such a limit, it treats larger texture maps in an implementation-dependent way. PEX implementations may have other restrictions on the dimensions of each level of the texture map. Inquire *PowerOfTwoTMRequired* with **PEXGetImpDepConstants** to determine if each dimension must be a power of two. Inquire *SquareTMRequired* with the same request to determine if all the dimensions on each level must be equal to one another.

Drawable indicates the visual of the resources. Both the *color_data* and *alpha_data* resources must be of the same visual class. The colormap associated with the drawable is used to determine the color values. The visual indicates the source texel format.

Not all resource types are supported for creating texture map resources. PEXlib defines the following texel resources. Invoke **PEXGetEnumTypeInfo** to determine which resources types your implementation supports.

> **PEXWindowResource** - The supplied resource is a mapped window. Treatment of obscured portions of the window is implementation-dependent if there is no backing store for the window. Consistent results require a completely unobscured window. (5.2)
>
> **PEXPixmapResource** - The supplied resource is a pixmap. (5.2)

PEXBufferResource - The supplied resource is a MBX buffer. If the buffer is currently displayed, treatment of obscured portions is implementation-dependent if there is no backing store for the buffer. Consistent results require a completely unobscured buffer. (5.2)

PEXPhotomapResource - The supplied resource is an XIE Photomap. (5.2)

Once this function returns, all texel data source resources may be reused for other purposes

Errors

BadAlloc	The implementation failed to allocate the resource necessary to complete request.
BadDrawable	*drawable* is invalid.
BadMatch	A specified resource does not contain luminance data appropriate to the texel type and channel.
BadMatch	A specified resource does not contain alpha data appropriate to the texel type and channel.
BadMatch	A specified resource is of an unsupported type.
BadMatch	A pixmap is not a valid example drawable for this request.
BadValue	The specified texture map dimension is not valid.
BadValue	The specified texture map texel type is not valid.
BadValue	The specified number of resources does not match the number of resources expected.
BadValue	The texture map size is required to be a power of two.
BadValue	The texture map size is required to be square.
BadValue	Luminance channel value or alpha channel value is invalid.

See Also

PEXCreateMipmapTM, **PEXFreeTextureMap**

PEXFreeTextureMap - Free Texture Map (5.2)

Synopsis

void **PEXFreeTextureMap**(Display **display*, `PEXTextureMap` *texture_map*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
texture_map	The resource ID of the texture map to be freed [§2.3.22, page 29].

Returns

None.

Description

Invoke **PEXFreeTextureMap** to delete the association between the resource ID and the texture map.

The texture map storage is freed when no other resource references it.

Errors

BadValue	*texture_map* is not a valid texture map ID.

See Also

PEXCreateMipmapTM, **PEXCreateMipmapTMFromResources**

16.4. Texture Mapping Utilities

PEXFreeMipmap - Frees Texture Resources

Synopsis

void **PEXFreeMipmap**(int *domain*, `PEXTMDomainData` **domain_data*, `PEXMipmapLevel` **texel_array*)

Arguments

domain	The texture map domain type
domain_data	A pointer to texture map domain information
texel_array	Texel data to be freed [§2.21.2, page 49].

Returns

None

Description

Invoke **PEXFreeMipmap** to free storage returned by **PEXGenerateMipmap** utilities.

Texel data created by these functions can be immediately freed by **PEXFreeMipmap** after using them with the **PEXCreateMipmapTM** function.

Errors

None.

See Also

PEXCreateMipmapTM, **PEXGenerateMipmap**, **PEXGenerateMipmapFromResources**

PEXGenerateMipmap - Creates a Prefiltered Texture Map from Base Map

Synopsis

int **PEXGenerateMipmap**(int *domain*, `PEXTMDomainData` **domain_data*, unsigned long int *power_of_two_tm_required*, unsigned long int *square_tm_required*, `PEXMipmapLevel` **base_map*, `PEXMipmapLevel` ***texel_array*)

Arguments

domain	The texture map domain type(**PEXTextureMapTypeMipmap**).
domain_data	A pointer to domain information specifying the type of texture map to be created and the type of texel data [§2.29.16, page 66].
power_of_two_tm_required	Indicates whether maps with dimensions of power of two is required (**True** or **False**).
square_tm_required	Indicates whether square maps is required (**True** or **False**).
base_map	The texel data of the desired filtered texture map's base level [§2.29.14, page 66].
texel_array	Texel data created by this utility [§2.29.14, page 66].

Returns

Zero if successful; otherwise, one of the following:

PEXBadDomain	Domain is not valid.
PEXBadDimension	Dimension is not valid.
PEXBadTexelType	Texel type is not valid.
PEXBadParamerization	Parameterization type is not valid.
PEXBadTMAlloc	Memory allocation error.
PEXBadTexelDimension	Texel dimensions are not valid.
PEXBadTexelData	Texel data is not valid.

Description

Invoke **PEXGenerateMipmap** to create a prefiltered texture map and get a list of texel arrays suitable for the **PEXCreateMipmapTM** function.

The information contained in *domain* and *domain_data* determines if a mipmap or other type of texture map is created from the base level data provided. At present, color mipmaps are the only type of domain supported. Subsequent releases of PEX may support others.

You need to set the *tm_type*, *texel_type*, and *num_levels* members of the `PEXTMDomainData` structure before invoking this function. A value of zero in *num_levels* indicates to the function that it should generate as many levels as necessary to create a full mipmap pyramid or other type of supported map. If the value is non-zero, then the utility creates either the number of levels requested or all the levels for a full map, whichever is less. Upon exit, the *num_levels* field is updated to the number of levels actually created by the utility.

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

Texel arrays allocated by this function are disposed of using the **PEXFreeMipmap** utility after they have been used by the **PEXCreateMipmapTM** function.

Errors

None.

See Also

PEXCreateMipmapTM, **PEXFreeMipmap**

PEXGenerateMipmapFromResources - Creates a Prefiltered Texture Map from Color and Alpha Resources

Synopsis

int **PEXGenerateMipmapFromResources**(Display **display*, int *domain*, `PEXTMDomainData` **domain_data*, int *resource_type*, `PEXTexelDimension` **texel_dimension*, unsigned long int *power_of_two_tm_required*, unsigned long int *square_tm_required*, int *luminance_channel*, XID *base_color_map*, XID *base_alpha_map*, `PEXMipmapLevel` ***texel_array*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
domain	The texture map domain type (**PEXTextureMapTypeMipmap**).
domain_data	A pointer to domain information specifying the type of texture map to be created and the type of texel data [§2.29.16, page 66].
resource_type	Type of resource specified by the XIDs (**PEXWindowDrawable, PEXPixmapDrawable** or **PEXBufferDrawable**)
texel_dimension	A pointer to a texture dimension structure that corresponds to the *base_color_map* [§2.29.13, page 65]
power_of_two_tm_required	Indicates whether maps with dimensions of power of two is required (**True** or **False**).
square_tm_required	Indicates whether square maps are required (**True** or **False**).
luminance_channel	Luminance channel source selector (**PEXLuminanceChannelLuminance, PEXLuminanceChannelRed, PEXLuminanceChannelGreen** or **PEXLuminanceChannelBlue**)
base_color_map	The X resource identifier of the desired filtered texture map's base level color and/or luminance data.
base_alpha_map	The X resource identifier of the desired filtered texture map's base level alpha data.
texel_array	Texel data created by this utility [§2.29.14, page 66].

Returns

Zero if successful; otherwise, one of the following:

PEXBadXResource	X resource ID is not valid.
PEXBadDomain	Domain is not valid.
PEXBadDimension	Dimension is not valid.
PEXBadTexelType	Texel type is not valid.
PEXBadParameterization	Parameterization type is not valid.
PEXBadTMAlloc	Memory allocation error.
PEXBadTexelDimension	Texel dimensions are not valid.
PEXBadTexelData	Texel data is not valid.

Description

Invoke **PEXGenerateMipmapFromResources** to create a prefiltered texture map and get texel data suitable for **PEXCreateMipmapTM**.

The information contained in *domain* and *domain_data* determines if a mipmap or other type of texture map is created from the base level data provided.

You need to set the *tm_type*, *texel_type*, and *num_levels* members of the `PEXTMDomainData` structure before invoking this function. A value of zero in *num_levels* indicates to the function that it should generate as many levels as necessary to create a full mipmap pyramid or other type of supported map. If the value is non-zero, then the utility creates either the number of levels requested or all the levels for a full map, whichever is less. Upon exit, the *num_levels* field is updated to the number of levels actually created by the utility.

You must specify the type of resource referenced by the XID for correct conversion by this utility. It is assumed that both the base color and alpha maps are of the same type. The current supported types are those specified by **PEXMatchRenderingTargets**.

If the texel type specified in the description indicates luminance is included (i.e., the texel type is (**PEXTexelTypeLum, PEXTexelTypeLum8, PEXTexelTypeLum16, PEXTexelTypeLumA, PEXTexelTypeLumA8, PEXTexelTypeLumA16**) then the *luminance_channel* indicates the source of the luminance data. The channels available as source of the luminance data are dependent upon the depth or visual class of the *base_color_map* resource. If the resource is a 24-bit resource, then red, green and blue channels are available as a source for luminance. If it is an 8-bit resource, then the whole 8-bit channel is used as the source for luminance.

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

Texel arrays allocated by this function are disposed of using the **PEXFreeMipmap** utility after they have been used by the **PEXCreateMipmapTM** function.

Errors

None.

See Also

PEXCreateMipmapTM, **PEXFreeMipmap**

PEXOCCTMCoordFillAreaSet - Computes a Projection of Texture Coordinates for Fill Area Set Primitives

Synopsis

int **PEXOCCTMCoordFillAreaSet**(PEXOCC *context*, `PEXTMCoordData` **tm_coord_data*, `PEXPointer` *facet_data*, unsigned int *count*, `PEXListOfVertexData` **vertex_lists*)

Arguments

context An OC context returned by a successful **PEXCreateOCC** call.

tm_coord_data A pointer to parameterization data for this primitive (See Description below) [§2.29.15, page 66].

facet_attributes(OCC)
A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).

surface_vertex_attributes(OCC)
A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGA2D** or **PEXGAFloatData**).

color_type(OCC) The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8** or **PEXColorTypeRGBA16**).

facet_fp_data_count (OCC)
The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count(OCC)
The number of floating-point data values defined with each vertex.

facet_data A pointer to the facet data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

count The number of fill areas in the set.

vertex_lists A pointer to the list of vertex arrays defining each contour of the fill area set [§2.21.5, page 50]. See "Facet / Vertex Data Formats" on page 137.

data_model (OCC) Specifies the data model used to represent the geometric data.

data_model_specs (OCC)
Offset and size data for the `PEXDataStride` or `PEXDataUnpacked` data models.

Returns

Zero if successful; otherwise, one of the following:

PEXBadTMPrimitive	Normals are not valid.
PEXBadValue	Projection type is not valid.
PEXBadValue	Texture coordinate location not valid.
PEXBadAlloc	Internal allocation error has occurred.

Description

Invoke **PEXOCCTMCoordFillAreaSet** to compute a projection of texture coordinates onto a fill area set and store them in the floating-point data areas of each vertex.

To compute texture coordinates, the following values in the *tm_coord_data* structure must be supplied by the client:

projection Texture coordinate projection to perform (**PEXTMProjectionSphereWC,**

	PEXTMProjectionCylinderWC or **PEXTMProjectionLinearWC**).
matrix	If projection is either **PEXTMProjectionSphereWC** or **PEXTMProjectionCylinderWC**, this transform is applied to vertices and vertex normals after the mc_transform has been applied. It is used to orient the data relative to the projection object (sphere or cylinder) as described.
p0, p1	If projection is **PEXTMProjectionLinearWC**, then these elements contain linear projection equations. (See description below.)
coord_source	Source coordinate to compute projections specified with projection type above (**PEXTMCoordSourceVertexCoord** or **PEXTMCoordSourceVertexNormal**).
fp_data_index	Index within the vertex floating-point data list in which to store the calculated texture coordinates. Space must already exist in the vertex floating-point data list and the index must point to a valid location. The projection generates a data list of two consecutive floating-point numbers.
mc_transform	This transform is applied to vertices and vertex normals in model coordinates prior to computing the specified projection.

Additionally, for the given projections, the following information must be provided. For projections listed below, those which require a unit direction vector (*s*0,*s*1,*s*2) shall have one determined by the coordinate source selection previously mentioned. If you select **PEXTMCoordSourceVertexCoord**, then a ray is computed from the WC origin through the vertex to the interior of the projection object. If you select **PEXTMCoordSourceVertexNormal**, then the primitive's vertex normals (given or implicit) are used to determine the direction. The resulting values (*t*0,*t*1) are the texture coordinates stored at the requested index.

PEXTMProjectionSphereWC

The infinite sphere has (0,0,0) as its origin and the +*y* axis as its axis of revolution. The texture seam sweeps from the +*x* axis in a counter-clockwise direction [0..2PI]. Given the specified coordinate source, a direction vector (*s*0,*s*1,*s*2) is computed to determine a point on the interior of the sphere. The resulting texture coordinates (*t*0,*t*1) are stored in the vertex's floating-point data list.

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

t1 = arcsin(s1) / PI + 0.5

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

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

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

PEXTMProjectionCylinderWC

The infinite cylinder has (0,0,0) as its origin and the +*y* axis as its axis of revolution. The texture seam sweeps from the +*x* axis in a counter-clockwise direction [0..2PI]. Given coordinate sources which are directional (normal or -reflection), a direction vector (*s*0,*s*1,*s*2) is computed to determine a point on the interior of the cylinder. If the coordinate source is vertex coord, a ray perpendicular to the +*y* axis through the vertex is computed to determine a point and its height on the interior of the cylinder. The resulting texture coordinates (*t*0,*t*1) are stored in the vertex's floating-point data list.

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

```
((s0 < 0) ? 0.5 : (s0 > 0 && s2 > 0) ? 1 : 0)
)
```

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

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

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

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

PEXExtTMProjectionLinearWC

Arguments for two linear equations are defined as *p*0 and *p*1. Given the specified coordinate source, the values are used directly as (*s*0,*s*1,*s*2) to linearly project the values by the equations listed below. The resulting texture coordinates (*t*0,*t*1) are stored in the vertex's floating-point data list. (where *s*3 is assumed to be one as shown in the equations below.)

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

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

The projection of anything other than vertices for this option produces unpredictable visual results.

It is assumed that the vertex list of the specified primitive has sufficient space to insert the needed texture coordinates. If space (indicated by **PEXGAFloatData**) or valid indices for texture coordinates do not exist, then the implementation issues an error and the data remains unchanged.

Errors

None.

See Also

PEXOCCTMCoordIndexedFillAreaSets, **PEXOCCTMCoordIndexedTriangles**, **PEXOCCTMCoordPolyTriangle**, **PEXOCCTMCoordQuadrilateralMesh**, **PEXOCCTMCoordTriangleFan**, **PEXOCCTMCoordTriangles**, **PEXOCCTMCoordTriangleStrip**

PEXOCCTMCoordIndexedFillAreaSets - Computes a Projection of Texture Coordinates for Indexed Fill Area Sets (SOFAS)

Synopsis

int **PEXOCCTMCoordIndexedFillAreaSets**(PEXOCC *context*, `PEXTMCoordData` **tm_coord_data*, unsigned int *set_count*, `PEXPointer` *facet_data*, unsigned int *vertex_count*, `PEXPointer` *vertices*, unsigned int *index_count*, `PEXSwitch` **edge_flags*, `PEXConnectivityData` **connectivity*)

Arguments

context — An OC context returned by a successful **PEXCreateOCC** call.

tm_coord_data — A pointer to parameterization data for this primitive (See Description in **PEXOCCTMCoordFillAreaSet**) [§2.29.17, page 66].

facet_attributes(OCC) — A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).

surface_vertex_attributes(OCC) — A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGA2D** or **PEXGAFloatData**).

color_type(OCC) — The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8** or **PEXColorTypeRGBA16**).

facet_fp_data_count (OCC) — The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count(OCC) — The number of floating-point data values defined with each vertex.

edge_attributes (OCC) — A mask indicating the edge attributes provided (**PEXGANone** or **PEXGAEdges**).

set_count — The number of fill area sets.

facet_data — A pointer to the facet data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

vertex_count — The number of vertices.

vertices — A pointer to the vertex data [§2.21.5, page 50]. See “Facet / Vertex Data Formats” on page 137.

index_count — The number of vertex connectivity indices (also number of edge flags, if edges are specified).

edge_flags — An array of edge flags [§2.3.20, page 29].

connectivity — A pointer to the list of contour connectivity data [§2.6.2, page 33].

data_model (OCC) — Specifies the data model used to represent the geometric data.

data_model_specs (OCC) — Offset and size data for the `PEXDataStride` or `PEXDataUnpacked` data models.

Returns

Same as for **PEXOCCTMCoordFillAreaSet**.

Description

Invoke **PEXOCCTMCoordIndexedFillAreaSets** to compute a projection of texture coordinates onto the indexed fill area sets and store them in the specified vertex data fields.

The rest of the description is identical to the description for **PEXOCCTMCoordFillAreaSet**.

Errors

None.

See Also

PEXOCCTMCoordFillAreaSet, **PEXOCCTMCoordIndexedTriangles**, **PEXOCCTMCoordPolyTriangle**, **PEXOCCTMCoordQuadrilateralMesh**, **PEXOCCTMCoordTriangleFan**, **PEXOCCTMCoordTriangles**, **PEXOCCTMCoordTriangleStrip**

PEXOCCTMCoordIndexedTriangles - Computes a Projection of Texture Coordinates for a Indexed Triangle Set

Synopsis

int **PEXOCCTMCoordIndexedTriangles**(PEXOCC *context*, `PEXTMCoordData` **tm_coord_data*, unsigned int *all_independent*, unsigned int *num_facets*, `PEXPointer` *facet_data*, unsigned int *num_verts*, `PEXPointer` *vertices*, `PEXSwitch` **per_edge_data*, unsigned short **connectivity*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
tm_coord_data	A pointer to parameterization data for this primitive (See Description in **PEXOCCTMCoordFillAreaSet**) [§2.29.17, page 66].
facet_attributes(OCC)	A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).
surface_vertex_attributes(OCC)	A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGA2D** or **PEXGAFloatData**).
edge_attributes (OCC)	A mask indicating the edge attributes provided (**PEXGANone**, **PEXGAEdges**).
color_type(OCC)	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8** or **PEXColorTypeRGBA16**).
facet_fp_data_count (OCC)	The number of optional floating-point numbers for each facet.
surface_vertex_fp_data_count(OCC)	The number of floating-point data values defined with each vertex.
all_independent	**True** if vertices are a simple list.
num_facets	Number of facets.
facet_data	A pointer to the facet data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.
num_verts	The number of vertices.
vertices	A pointer to the vertex data [§2.21.5, page 50]. See “Facet / Vertex Data Formats” on page 137.
per_edge_data	Optional edge data [§2.3.20, page 29].
connectivity	A pointer to the list of triangle connectivity data.
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the `PEXDataStride` or `PEXDataUnpacked` data models.

Returns

Same as for **PEXOCCTMCoordFillAreaSet**.

Description

Invoke **PEXOCCTMCoordIndexedTriangles** to compute a projection of texture coordinates onto the indexed triangles and store them in the specified vertex data fields.

The rest of the description is identical to the description for **PEXOCCTMCoordFillAreaSet**.

Errors

None.

See Also

PEXOCCTMCoordFillAreaSet, PEXOCCTMCoordIndexedFillAreaSets, **PEXOCCTMCoordPolyTriangle**, **PEXOCCTMCoordQuadrilateralMesh**, **PEXOCCTMCoordTriangleFan**, **PEXOCCTMCoordTriangles**, **PEXOCCTMCoordTriangleStrip**

PEXOCCTMCoordPolyTriangle - Computes a Projection of Texture Coordinates for a PolyTriangle

Synopsis

int **PEXOCCTMCoordPolyTriangle**(PEXOCC *context*, `PEXTMCoordData` **tm_coord_data*, `PEXPointer` *facet_data*, unsigned int *count*, `PEXPointer` *vertices*)

Arguments

context — An OC context returned by a successful **PEXCreateOCC** call.

tm_coord_data — A pointer to parameterization data for this primitive (See Description in **PEXOCCTMCoordFillAreaSet**) [§2.29.17, page 66].

facet_attributes(OCC) — A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).

surface_vertex_attributes(OCC) — A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGA2D** or **PEXGAFloatData**).

color_type(OCC) — The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8** or **PEXColorTypeRGBA16**).

facet_fp_data_count (OCC) — The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count(OCC) — The number of floating-point data values defined with each vertex.

facet_data — A pointer to the facet data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

count — The number of vertices.

vertices — A pointer to the vertex data [§2.21.5, page 50]. See “Facet / Vertex Data Formats” on page 137.

data_model (OCC) — Specifies the data model used to represent the geometric data.

data_model_specs (OCC) — Offset and size data for the `PEXDataStride` or `PEXDataUnpacked` data models.

Returns

Same as for **PEXOCCTMCoordFillAreaSet**.

Description

Invoke **PEXOCCTMCoordPolyTriangle** to compute a projection of texture coordinates onto a poly-triangle and store them in the specified vertex data fields.

The rest of the description is identical to the description for **PEXOCCTMCoordFillAreaSet**.

Errors

None.

See Also

PEXOCCTMCoordFillAreaSet, PEXOCCTMCoordIndexedFillAreaSets, **PEXOCCTMCoordIndexedTriangles**, **PEXOCCTMCoordQuadrilateralMesh**, **PEXOCCTMCoordTriangleFan**, **PEXOCCTMCoordTriangles**, **PEXOCCTMCoordTriangleStrip**

PEXOCCTMCoordQuadrilateralMesh - Computes a Projection of Texture Coordinates for a Quadrilateral Mesh

Synopsis

int **PEXOCCTMCoordQuadrilateralMesh**(PEXOCC *context*, `PEXTMCoordData` **tm_coord_data*, `PEXPointer` *facet_data*, unsigned int *col_count*, unsigned int *row_count*, `PEXPointer` *vertices*)

Arguments

context — An OC context returned by a successful **PEXCreateOCC** call.

tm_coord_data — A pointer to parameterization data for this primitive (See Description in **PEXOCCTMCoordFillAreaSet**) [§2.29.17, page 66].

facet_attributes(OCC) — A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).

surface_vertex_attributes(OCC) — A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGA2D** or **PEXGAFloatData**).

color_type(OCC) — The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8** or **PEXColorTypeRGBA16**).

facet_fp_data_count (OCC) — The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count(OCC) — The number of floating-point data values defined with each vertex.

facet_data — A pointer to the facet data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

col_count — The number of columns in the vertex array.

row_count — The number of rows in the vertex array.

vertices — A pointer to the vertex data interpreted as a two-dimensional (row-major) array of vertices defining the quadrilateral mesh [§2.21.5, page 50]. See “Facet / Vertex Data Formats” on page 137.

data_model (OCC) — Specifies the data model used to represent the geometric data.

data_model_specs (OCC) — Offset and size data for the `PEXDataStride` or `PEXDataUnpacked` data models.

Returns

<Same as for **PEXOCCTMCoordFillAreaSet**.>

Description

Invoke **PEXOCCTMCoordQuadrilateralMesh** to compute a projection of texture coordinates onto a quadrilateral mesh and store them in the specified vertex data fields.

The rest of the description is identical to the description for **PEXOCCTMCoordFillAreaSet**.

Errors

None.

See Also

PEXOCCTMCoordFillAreaSet, **PEXOCCTMCoordIndexedFillAreaSets**, **PEXOCCTMCoordIndexedTriangles**, **PEXOCCTMCoordPolyTriangle**, **PEXOCCTMCoordTriangleFan**, **PEXOCCTMCoordTriangles**, **PEXOCCTMCoordTriangleStrip**

PEXOCCTMCoordTriangleFan - Computes a Projection of Texture Coordinates for a Triangle Fan

Synopsis

int **PEXOCCTMCoordTriangleFan**(PEXOCC *context*, `PEXTMCoordData` **tm_coord_data*, `PEXPointer` *facet_data*, unsigned int *count*, `PEXPointer` *vertices*)

Arguments

context — An OC context returned by a successful **PEXCreateOCC** call.

tm_coord_data — A pointer to parameterization data for this primitive (See Description in **PEXOCCTMCoordFillAreaSet**) [§2.29.17, page 66].

facet_attributes(OCC) — A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).

surface_vertex_attributes(OCC) — A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGA2D** or **PEXGAFloatData**).

color_type(OCC) — The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8** or **PEXColorTypeRGBA16**).

facet_fp_data_count (OCC) — The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count(OCC) — The number of floating-point data values defined with each vertex.

facet_data — A pointer to the facet data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

count — The number of vertices.

vertices — A pointer to the vertex data [§2.21.5, page 50]. See "Facet / Vertex Data Formats" on page 137.

data_model (OCC) — Specifies the data model used to represent the geometric data.

data_model_specs (OCC) — Offset and size data for the `PEXDataStride` or `PEXDataUnpacked` data models.

Returns

Same as for **PEXOCCTMCoordFillAreaSet**.

Description

Invoke **PEXOCCTMCoordTriangleFan** to compute a projection of texture coordinates onto a triangle fan and store them in the specified vertex data fields.

The rest of the description is identical to the description for **PEXTMCoordFillAreaSetWithData**.

Errors

None.

See Also

PEXOCCTMCoordFillAreaSet, **PEXOCCTMCoordIndexedFillAreaSets**, **PEXOCCTMCoordIndexedTriangles**, **PEXOCCTMCoordPolyTriangle,** **PEXOCCTMCoordQuadrilateralMesh**, **PEXOCCTMCoordTriangleStrip**, **PEXOCCTMTriangles**

PEXOCCTMCoordTriangles - Computes a Projection of Texture Coordinates for a List of Independent Triangles

Synopsis

int **PEXOCCTMCoordTriangles**(PEXOCC *context*, `PEXTMCoordData` **tm_coord_data*, `PEXPointer` *facet_data*, unsigned int *count*, `PEXPointer` *vertices*)

Arguments

context — An OC context returned by a successful **PEXCreateOCC** call.

tm_coord_data — A pointer to parameterization data for this primitive (See Description in **PEXOCCTMCoordFillAreaSet**) [§2.29.17, page 66].

facet_attributes(OCC) — A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).

surface_vertex_attributes(OCC) — A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGA2D** or **PEXGAFloatData**).

color_type(OCC) — The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8** or **PEXColorTypeRGBA16**).

facet_fp_data_count (OCC) — The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC) — The number of floating-point data values defined with each vertex.

facet_data — A pointer to the facet data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

count — The number of vertices.

vertices — A pointer to the vertex data [§2.21.5, page 50]. See “Facet / Vertex Data Formats” on page 137.

data_model (OCC) — Specifies the data model used to represent the geometric data.

data_model_specs (OCC) — Offset and size data for the `PEXDataStride` or `PEXDataUnpacked` data models.

Returns

Same as for **PEXOCCTMCoordFillAreaSet**.

Description

Invoke **PEXOCCTMCoordTriangles** to compute a projection of texture coordinates onto a list of independent triangles and store them in the specified vertex data fields.

The rest of the description is identical to the description for **PEXOCCTMCoordFillAreaSet**.

Errors

None.

See Also

PEXOCCTMCoordFillAreaSet, PEXOCCTMCoordIndexedFillAreaSets, **PEXOCCTMCoordIndexedTriangles**, **PEXOCCTMCoordPolyTriangle, PEXOCCTMCoordQuadrilateralMesh**, **PEXOCCTMCoordTriangleFan**, **PEXOCCTMCoordTriangleStrip**

PEXOCCTMCoordTriangleStrip - Computes a Projection of Texture Coordinates for a Triangle Strip

Synopsis

int **PEXOCCTMCoordTriangleStrip**(PEXOCC *context*, `PEXTMCoordData` **tm_coord_data*, `PEXPointer` *facet_data*, unsigned int *count*, `PEXPointer` *vertices*)

Arguments

context — An OC context returned by a successful **PEXCreateOCC** call.

tm_coord_data — A pointer to parameterization data for this primitive (See Description in **PEXOCCTMCoordFillAreaSet**) [§2.29.17, page 66].

facet_attributes(OCC) — A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal** or **PEXGAFloatData**).

surface_vertex_attributes(OCC) — A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**, **PEXGAEdges**, **PEXGA2D** or **PEXGAFloatData**).

color_type(OCC) — The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**, **PEXColorTypeRGBA**, **PEXColorTypeRGBA8** or **PEXColorTypeRGBA16**).

facet_fp_data_count (OCC) — The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC) — The number of floating-point data values defined with each vertex.

facet_data — A pointer to the facet data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

count — The number of vertices.

vertices — A pointer to the vertex data [§2.21.5, page 50]. See “Facet / Vertex Data Formats” on page 137.

data_model (OCC) — Specifies the data model used to represent the geometric data.

data_model_specs (OCC) — Offset and size data for the `PEXDataStride` or `PEXDataUnpacked` data models.

Returns

Same as for **PEXOCCTMCoordFillAreaSet**.

Description

Invoke **PEXOCCTMCoordTriangleStrip** to compute a projection of texture coordinates onto a triangle strip and store them in the specified vertex data fields.

The rest of the description is identical to the description for **PEXOCCTMCoordFillAreaSet**.

Errors

None.

See Also

PEXOCCTMCoordFillAreaSet, **PEXOCCTMCoordIndexedFillAreaSets**, **PEXOCCTMCoordIndexedTriangles**, **PEXOCCTMCoordPolyTriangle,** **PEXOCCTMCoordQuadrilateralMesh**, **PEXOCCTMCoordTriangleFan**, **PEXOCCTMTriangles**

Chapter 17

Accumulation Buffers

The *accumulation buffer* is an ancillary buffer useful in advanced rendering techniques that generally require rendering the entire scene, or some portion of it, multiple times and “accumulating” the resulting images. Such techniques include, but need not be limited to: antialiasing, motion blur, depth of field, and transparency. For more information on techniques which take advantage of the accumulation buffer, see the paper, “The Accumulation Buffer: Hardware Support for High-Quality Rendering”, Computer Graphics, Volume 24, Number 4, August, 1990, pp. 309-318, written by Paul Haeberli and Kurt Akeley.

Accumulation is performed by adding the contents of an image (and possibly ancillary components of an image such as alpha values) to the buffer. Accumulation occurs using one of the following techniques, or by some combination:

- The images being added have a weighting applied to them at the time of accumulation
- the weighting of the contents of the accumulation buffer are adjusted during accumulation
- the resulting accumulation of all the images are averaged at the time the accumulation buffer’s contents are returned

At the end of accumulation, the buffer is typically transferred to a drawable to display the resulting image.

Accumulation buffers are not drawables; renderers cannot render directly to accumulation buffers. The images used during accumulation may consist of pixels that are the result from PEX rendering, so that in fact the accumulation buffers are “indirectly rendered”. Both the accumulation buffers and the accumulated images must contain colors which are the result of the application of the color approximation steps. Therefore, accumulation is defined only for component visuals, i.e. those visuals which have X11 pixel values that directly specify each of the red, green and blue components. See the definition of *DirectColor* and *TrueColor* visuals in the core X11 specification for more detail on visual types.

Since accumulation is performed on images, the accumulation buffer is originally loaded from a drawable and has drawables accumulated into it. Core X11 semantics provide no mechanism to specify the existence of ancillary buffers such as the accumulation buffer or alpha buffer. These ancillary buffers may be described by the function “PEXGetAncillaryBufferInfo - Get Ancillary Buffer Information (5.2)” on page 97, which the client uses to determine capabilities of a drawable. If the implementation supports drawables with an alpha buffer capability, then a PEX implementation supporting accumulation buffers must support the drawables with sufficient alpha planes to allow for accumulation of the ancillary alpha values. Additionally, the total depth of the accumulation buffer must be equal to or greater than the total depth of the drawable to which it is associated, and the depth of each component (red, green, blue, and if supported, alpha) must also be equal to or greater than the depth of each component. Also, since the accumulation buffer is an ancillary buffer to the drawable, it has the same width and height as the drawable.

If the drawable with which an accumulation buffer is associated is resized, all subsequent **PEXAccumulateBuffer** and **PEXReturnAccumulationBuffer** requests are ignored until the client makes a **PEXLoadAccumulationBuffer** or **PEXFreeAccumulationBuffer** request. The ignored accumulation buffer requests cause no further action and generate no errors. However, a **PEXLoadAccumulationBuffer** request may generate a *BadAlloc* error if the implementation attempts to resize the accumulation buffer due to its associated drawable being resized.

If the drawable with which an accumulation buffer is associated is destroyed, the implementation performs an implicit **PEXFreeAccumulationBuffer**.

If the drawable associatted with an accumulation buffer is moved, exposed, or occuluded, the implementation continues processing subsequent accumulation buffer requests using the newly modified drawable.

Typical methods of using the accumulation buffer are as follows:

1) Accumulate source images with source weights between 0.0 and 1.0, a destination weight of 1.0, and return without scaling. One application of this approach is for the blending of images into an accumulation buffer which has a depth

that is not sufficiently deeper than the source image to allow for overflow, also known as pixel saturation, on addition. The final image returned is the non-normalized result of the multiple accumulations.

2) Accumulate source images with source and destination weights of 1.0, and return after scaling by $1/n$, where n is the number of accumulations. For sufficiently deep accumulation buffers where overflow is generally not a problem, the image is added to the accumulation buffer. When the image is returned, it is averaged by the number of accumulations. Some implementations may recognize the 1.0 source weight and skip the multiplication on each accumulation of each pixel.

3) If the accumulation buffer is not sufficiently deep to avoid the possibility of overflow, and the number of accumulations is not known until all the accumulations are complete, then it is possible to adjust the running average by using both the source and destination weight factors. The return is then done without scaling.

PEXAccumulateBuffer - Accumulate Buffer (5.2)

Synopsis

void **PEXAccumulateBuffer**(Display **display*, Drawable *drawable*, float *source_weight*, float *dest_weight*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	A drawable with an accumulation buffer associated with it.
source_weight	Source weight (see the equation below).
dest_weight	Destination weight (see the equation below).

Returns

None.

Description

Invoke **PEXAccumulateBuffer** to accumulate the contents of *drawable (D)* into the ancillary accumulation buffer *(A)* associated with *drawable* using the *source_weight* (w_s) and *dest_weight* (w_d) as follows:

$$A_{x,y} = A_{x,y} w_d + D_{x,y} w_s$$

If underflow or overflow occurs, then the result is implementation-dependent.

If the *drawable* has been resized but its accumulation buffer has not yet been resized by using the **PEXLoadAccumulationBuffer** function, the implementation will take no action for this function.

Errors

BadDrawable	*drawable* does not specify a valid drawable.
BadMatch	The drawable specified by *drawable* does not have an allocated accumulation buffer.

See Also

PEXAllocateAccumulationBuffer, **PEXReturnAccumulationBuffer**

PEXAllocateAccumulationBuffer - Allocate Accumulation Buffer (5.2)

Synopsis

void **PEXAllocateAccumulationBuffer**(Display **display*, Drawable *drawable*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The accumulation buffer is associated with this drawable.

Returns

None.

Description

Invoke **PEXAllocateAccumulationBuffer** to allocate an accumulation buffer for the specified *drawable*.

The width and height of the accumulation buffer match that of the drawable. The depth of the buffer is implementation-dependent but must be at least as deep as the drawable. The initial contents of the buffer are undefined. The function **PEXLoadAccumulationBuffer** initializes the accumulation buffer.

Errors

BadAlloc	The implementation was unable to allocate the necessary resources.
BadMatch	*drawable* is invalid or does not specify a drawable capable of supporting an accumulation buffer.

See Also

PEXFreeAccumulationBuffer

PEXFreeAccumulationBuffer - Free Accumulation Buffer (5.2)

Synopsis

void **PEXFreeAccumulationBuffer**(Display **display*, Drawable *drawable*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The drawable associated with the accumulation buffer to free.

Returns

None.

Description

Invoke **PEXFreeAccumulationBuffer** to free the accumulation buffer associated with the specified *drawable*.

If no accumulation buffer is allocated for this drawable, then the implementation takes no further action.

Errors

BadDrawable	*drawable* is invalid.

See Also

PEXAllocateAccumulationBuffer

PEXLoadAccumulationBuffer - Load Accumulation Buffer (5.2)

Synopsis

void **PEXLoadAccumulationBuffer**(Display **display*, Drawable *drawable*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The drawable associated with the accumulation buffer to load.

Returns

None.

Description

Invoke **PEXLoadAccumulationBuffer** to copy the contents of the drawable into the accumulation buffer.

This function overwrites the existing values in that accumulation buffer.

Errors

BadAlloc	The implementation was unable to reallocate an accumulation buffer to match the resized *drawable*.
BadDrawable	*drawable* is invalid.
BadMatch	The drawable specified by *drawabl*e does not have an allocated accumulation buffer.

See Also

PEXAccumulateBuffer, **PEXAllocateAccumulationBuffer**

PEXReturnAccumulationBuffer - Return Accumulation Buffer (5.2)

Synopsis

void **PEXReturnAccumulationBuffer**(Display **display*, Drawable *drawable*, float *scale_factor*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The drawable which is loaded with the scaled contents of the accumulation buffer associated with it.
scale_factor	The scale factor used to scale the accumulated values as they are copied to the drawable.

Returns

None.

Description

Invoke **PEXReturnAccumulationBuffer** to multiply the contents of the accumulation buffer *(A)* associated with *drawable (D)* by the *scale_factor (w)* and then write to the drawable itself.

This function overwrites the existing values in the drawable. The operation is as follows:

$$D_{x,y} = A_{x,y}w$$

If underflow or overflow occurs, then the result is implementation-dependent.

If the *drawable* has been resized but its accumulation buffer has not yet been resized by using the **PEXLoadAccumulationBuffer** function, the implementation will take no action for this function.

Errors

BadDrawable	*drawable* is invalid.
BadMatch	The drawable specified by *drawable* does not have an allocated accumulation buffer.

See Also

PEXAccumulateBuffer

Chapter 18

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 aperture is also specified. The first output primitive that satisfies the filter criteria and is within the search aperture is returned.

18.1. Search Context

A *search context* is a PEX resource that contains all of 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 remain, because they are still referenced by the search context. However, if the search context is queried, the value **PEXAlreadyFreed** is returned for any such freed name sets, because they no longer have valid resource identifiers by which they can be referenced.

A search context resource contains the following attributes. All attributes became available with PEX 5.0, except *text font table*, which became available in PEX 5.2:

> *search position* - specifies the search reference position in world coordinates.
>
> *search distance* - specifies a distance from the search reference position in world coordinates. The search position and distance specify 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.
>
> *start path* - 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.
>
> *search ceiling* - 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.
>
> *model clip flag* - specifies whether model 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.
>
> *normal list* - 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.
>
> *inverted list* - 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.
>
> *text font table* - A resource identifier of the text font table which is used to resolve references by Text Font output commands. When the *text font table* is Null, the font assumed by the implementation is implementation-dependent. If you wish to reference no text font table, use the null resource identifier **None**.(PEX 5.2)

Use the **PEXSearchNetwork** function 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 within 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, then the search is unsuccessful, NULL is returned as the found path, and the start path attribute is not changed.

PEXChangeSearchContext5_2 - Change Search Context (5.0)

Synopsis

void **PEXChangeSearchContext5_2**(Display **display*, `PEXSearchContext` *context*, unsigned long *value_mask*, `PEXSCAttributes5_2` **values*)

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 [§2.3.18, page 29].
value_mask	A mask indicating the search context attributes to be changed.
values	A pointer to new values for the specified search context attributes [§2.27.2, page 63], [§2.27.1, page 63].

Returns

None.

Description

Invoke **PEXChangeSearchContext5_2** to modify the attributes of the specified search *context* resource. You must use the 5.2 form of the function unless all aspects of the components you wish to change were defined in PEX 5.0 or 5.1. Note that each of the two forms uses a different structure for the *values* parameter.

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
PEXSCTextFontTable

All attributes are supported beginning with PEX 5.0, except **PEXSCTextFontTable**, which is supported beginning with PEX 5.2.

Errors

BadPEXLookupTable	The specified text font lookup table is invalid
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 (5.0)

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 [§2.3.18, page 29].
dst_context	The resource identifier of the destination search context [§2.3.18, page 29].

Returns

None.

Description

Invoke **PEXCopySearchContext** to copy attributes from the source search context (*src_context*) to the destination search context (*dst_context*) resource.

Both the source and destination context must already exist as valid search context resources. This function copies attributes indicated by the *value mask* 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
PEXSCTextFontTable

All attributes are supported beginning with PEX 5.0, except **PEXSCTextFontTable**, which is supported beginning with PEX 5.2.

Errors

BadPEXSearchContext	A specified search context resource identifier is invalid.
BadValue	An invalid bit is set in the value mask.

See Also

PEXCreateSearchContext

PEXCreateSearchContext5_2 - Create Search Context (5.0)

Synopsis

`PEXSearchContext` **PEXCreateSearchContext5_2**(Display **display*, unsigned long *value_mask*,
    `PEXSCAttributes5_2` **values*)

`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 [§2.27.2, page 63], [§2.27.1, page 63].

Returns

The resource identifier of the newly-created search context resource [§2.3.18, page 29].

Description

Invoke **PEXCreateSearchContext5_2** to create a search context and get the resource identifier of the created search context resource. You must use the 5.2 form of the function unless all aspects of the components you wish to supply were defined in PEX 5.0 or 5.1. Note that each of the two forms uses a different structure for the *values* parameter.

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
PEXSCTextFontTable

All attributes are supported beginning with PEX 5.0, except **PEXSCTextFontTable**, which is supported beginning with PEX 5.2.

Errors

BadAlloc	The implementation failed to allocate the resource.
BadPEXLookupTable	The specified text font lookup table is invalid
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

PEXFreeSCAttributes5_2 - Free Storage Returned by PEXGetSearchContext (5.0)

Synopsis

void **PEXFreeSCAttributes5_2**(`PEXSCAttributes5_2` **values*)

void **PEXFreeSCAttributes**(`PEXSCAttributes` **values*)

Arguments

values A pointer to the search context attribute values [§2.27.2, page 63], [§2.27.1, page 63].

Returns

None.

Description

Invoke **PEXFreeSCAttributes5_2** to deallocate memory returned by **PEXGetSearchContext**. Be sure to use the correct form of the function depending on the type of the *values* parameter.

Errors

None.

See Also

PEXGetSearchContext

PEXFreeSearchContext - Free Search Context (5.0)

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 [§2.3.18, page 29].

Returns

None.

Description

Invoke **PEXFreeSearchContext** to delete the search context resource and free the memory associated with it.

Errors

BadPEXSearchContext	The specified search context resource identifier is invalid.

See Also

PEXCreateSearchContext

PEXGetSearchContext5_2 - Get Search Context Attributes (5.0)

Synopsis

`PEXSCAttributes5_2` ***PEXGetSearchContext5_2**(Display **display*, `PEXSearchContext` *context*, unsigned long *value_mask*)

`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 [§2.3.18, page 29].
value_mask	A mask indicating which attributes to return.

Returns

A pointer to the requested attribute values; a null pointer if unsuccessful [§2.27.2, page 63], [§2.27.1, page 63].

Description

Invoke **PEXGetSearchContext5_2** to get the requested attribute values of the specified search *context* resource. You must use the 5.2 form of the function unless all aspects of the components you wish to get were defined in PEX 5.0 or 5.1. Note that each of the two forms uses a different structure for the *values* parameter.

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
PEXSCTextFontTable

PEXlib allocates memory for the returned search context attribute values. Invoke **PEXFreeSCAttributes** to deallocate the memory.

All attributes are supported beginning with PEX 5.0, except **PEXSCTextFontTable**, which is supported beginning with PEX 5.2.

Errors

BadPEXSearchContext	The specified search context resource identifier is invalid.
BadValue	An invalid bit is set in the value mask.

See Also

PEXChangeSearchContext, **PEXCreateSearchContext**, **PEXFreeSCAttributes**

PEXSearchNetwork - Search Network (5.0)

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 [§2.3.18, page 29].
path_return	Returns a pointer to a structure network path identifying the first primitive found [§2.28.4, page 64].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXSearchNetwork** to search 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, then a null pointer is returned and the search context's start path is set to Null. If the primitive is found in a locked structure, the returned path is undefined.

After the search has been completed, the start path attribute of the specified search context is set to the path returned, if a primitive was found. If the stucture is locked, the first valid position in the structure network beyond the primitive which met the search criteria is used to set the return path A valid position is the next structure referencing output command in the locked structure, or the reference to the locked structure, unless the end of the search occurs before encountering the next valid position.

PEXlib allocates memory for the returned structure path. Invoke **PEXFreeStructurePaths** to deallocate the memory.

Errors

BadPEXPath	The specified path is invalid.
BadPEXSearchContext	The specified search context resource identifier is invalid.
BadPEXStructurePermission	The specified path includes an invalid position in a locked structure.

See Also

PEXChangeSearchContext, **PEXCreateSearchContext**, **PEXFreeStructurePaths**

Chapter 19

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.

The PHIGS workstation resource is maintained in PEX 5.2 and PEXlib 5.2 solely for backwards-compatiblity reasons. No new PEX 5.2 functionality is added to the workstation resource. For example, the PEX 5.2 testure mapping lookup table has not been added to the list of attributes defined below. However, some PEX 5.2 functionality may still be usable from the workstation subset, such as new output commands that do not implicitly reference new lookup tables or other new resources.

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

You can inquire these attributes but not modify them during the life of the PHIGS workstation. These attributes function in a way identical to their counterparts in a *renderer resource*. There is no view table as in the renderer. The workstation has a built-in view table that you can access only through the defined workstation functions. However, this view table is functionally the same as that used by the renderer.

19.1. Modifying PHIGS Workstation Attributes

A workstation contains the following attributes that can be modified or inquired with various functions:

19.1.1. Posted Structure Networks

The posted structure network attributes 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.

Invoke the **PEXPostStructure** function to post structure networks to a PHIGS workstation resource. Invoke the **PEXUnpostStructure** function to unpost a structure network. Invoke the **PEXUnpostAllStructures** function to reset the posted structure network list to the null list. Finally, given a structure resource id, invoke the **PEXGetWorkstationPostings** function to get a list of all PHIGS workstations to which the structure has been posted.

19.1.2. Requested and Current Values

The PHIGS workstation resource supports the concepts of *requested* changes and *current* values. You 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 you call 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:

requested NPC subvolume	current NPC subvolume
requested viewport	current viewport
requested HLHSR mode	current HLHSR mode
requested buffer mode	current buffer mode

19.1.3. Defined Views

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.

Use the **PEXSetWorkstationViewPriority** function to adjust the view priorities. 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 when you invoke the **PEXExecuteDeferredActions** function.

19.1.4. Display Update Mode

The display update mode indicates how the PHIGS workstation should attempt to visualize changes that occur. PEXlib defines the following types:

PEXVisualizeEach - displays each change as it occurs, even doing a retraversal of the posted structure network list if necessary.

PEXVisualizeEasy - display changes, as they occur, 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.

PEXVisualizeNone - applies changes to the structure, but the structure is not traversed until the application program explicitly requests it.

PEXVisualizeWhenever - traversal the structure until it is done when it is convenient to do so.

PEXSimulateSome - displays easy or fast changes immediately and simulates others by such means as drawing a deleted primitive in the background color.

19.2. Maintaining the State of the PHIGS Workstation

A PHIGS workstation contains the following attributes to maintain the state of the PHIGS workstation resource:

visual state - 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.

display surface - indicates whether the PHIGS workstation has rendered anything on the associated drawable.

update attributes - these include:

view update
workstation update
HLHSR update
buffer update

and 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.

number of priorities - indicates the PHIGS workstation's priorities.

You can inquire these attributes but not modify them: they are modified as side effects of various requests.

Invoke the **PEXRedrawAllStructures** function to explicitly retraverse a PHIGS workstation's posted structure networks. Invoke the **PEXUpdateWorkstation** function to perform a traversal if the visual state is not correct. Invoke the **PEXGetWorkstationDynamics** function 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 **PEXCorrect** at the end of the traversal.

Pick device descriptors are also a part of the workstation resource. They are defined in *Chapter 20:PHIGS Workstation Picking*.

If the drawable associated with a workstation resource is destroyed, then the workstation resource is implicitly freed. The next access to the workstation generates a *BadPEXWorkstation* error.

Lookup tables and name set resource can be bound to a workstation resource and then freed. The contents of these resources remain, because they are still being referenced by the workstation. However, when you inquire a workstation's attributes, the value **PEXAlreadyFreed** is 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 [§2.3.8, page 28].
marker_bundle	The resource identifier of the marker bundle lookup table [§2.3.8, page 28].
text_bundle	The resource identifier of the text bundle lookup table [§2.3.8, page 28].
interior_bundle	The resource identifier of the interior bundle lookup table [§2.3.8, page 28].
edge_bundle	The resource identifier of the edge bundle lookup table [§2.3.8, page 28].
color_table	The resource identifier of the color lookup table [§2.3.8, page 28].
pattern_table	The resource identifier of the pattern lookup table [§2.3.8, page 28].
font_table	The resource identifier of the text font lookup table [§2.3.8, page 28].
depth_cue_table	The resource identifier of the depth cue lookup table [§2.3.8, page 28].
light_table	The resource identifier of the light lookup table [§2.3.8, page 28].
color_approx_table	The resource identifier of the color approximation lookup table [§2.3.8, page 28].
highlight_incl	The name set used as the highlight inclusion set [§2.3.13, page 28].
highlight_excl	The name set used as the highlight exclusion set [§2.3.13, page 28].
invisibility_incl	The name set used as the invisibility inclusion set [§2.3.13, page 28].
invisibility_excl	The name set used as the invisibility exclusion set [§2.3.13, page 28].
buffer_mode	The workstation buffering mode (**PEXSingleBuffer** or **PEXDoubleBuffer**).

Returns

The resource identifier of the newly-created workstation [§2.3.24, page 29].

Description

Invoke **PEXCreateWorkstation** to create a PHIGS workstation resource and get its resource identifier.

The specified *drawable* 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. To modify or inquire the PHIGS workstation view table, invoke the **PEXSetWorkstationViewRep** and **PEXGetWorkstationViewRep** requests.

If the workstation is to operate in double-buffered mode and the implementation supports double-buffering, then an additional image buffer is created for the drawable in an implementation-dependent way. If you specify a drawable that is a pixmap, then no double-buffering is performed. To determine if your implementation supports double-buffering, invoke the **PEXGetImpDepConstants** function request.

Errors

BadAlloc	The implementation failed to allocate the resource, or the implementation 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 [§2.3.24, page 29].

Returns

None.

Description

Invoke **PEXExecuteDeferredActions** to execute all deferred actions on the specified *workstation*.

This function makes current all requested attributes and sets the corresponding update attributes to **PEXNotPending**.

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 [§2.3.24, page 29].

Returns

None.

Description

Invoke **PEXFreeWorkstation** to delete the PHIGS workstation resource and free 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 [§2.23.7, page 53].

Returns

None.

Description

Invoke **PEXFreeWorkstationAttributes** to deallocate memory returned by the **PEXGetWorkstationAttributes** function.

Errors

None.

See Also

PEXGetWorkstationAttributes

PEXGetWorkstationDynamics - Get Workstation Attribute Modification Dynamics

Synopsis

Status **PEXGetWorkstationDynamics**(Display **display*, Drawable *drawable*,
`PEXWorkstationDynamics` **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 [§2.23.8, page 54].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetWorkstationDynamics** to get the dynamics of the specified PHIGS workstation resource.

Each entry in the returned list has one of the following values:

PEXIMM- modifying the corresponding attribute immediately displays the correct image without regeneration.

PEXCBS - modifying the corresponding attribute immediately displays a simulation if you set the display update mode to **PEXSimulateSome.**

PEXIRG - requires a regeneration for the change to be visible.

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 [§2.3.24, page 29].
value_mask	A mask indicating which workstation attribute values to return.

Returns

A pointer to the workstation attribute values; a null pointer if unsuccessful [§2.23.7, page 53].

Description

Invoke PEXGetWorkstationAttributes to the attributes of the specified *workstation*.

This function returns the attributes specified by the *value mask*. Use the **PEXSetPWAttributeMask** or **PEXSetPWAttributeMaskAll** functions to setup the *value mask*.

PEXlib allocates memory for the returned workstation attribute values. Invoke **PEXFreeWorkstationAttributes** to deallocate the memory.

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 [§2.3.19, page 29].
count_return	Returns the number of workstation identifiers.
postings_return	Returns an array of workstation resource identifiers [§2.3.24, page 29].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetWorkstationPostings** to get a list of workstation resources to which the specified *structure* has been posted.

PEXlib allocates memory for the return value. Invoke **XFree** to deallocate the memory.

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 [§2.3.24, page 29].
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 [§2.23.6, page 53].
cur_view_return	Returns the current value of the view table index [§2.23.6, page 53].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXGetWorkstationViewRep** to get the update state (*update_return*), and the requested (*req_view_return*) and current values (*cur_view_return*) for the specified view *index* of the specified *workstation*.

The update is **PEXPending** if a view change has been requested but not established; otherwise the update is **PEXNotPending**. If the specified view index is not defined, then the implementationimplementation issues a *BadValue* error and the contents of the return arguments are undefined.

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 [§2.3.24, page 29].
dc_count	The number of device coordinate points.
dc_points	An array of device coordinates [§2.26.3, page 62].
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 [§2.7.1, page 33].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke **PEXMapDCToWC** to map the device coordinate points to world coordinate points using the specified *workstation*.

This function checks each view in the workstation's current view table, which has an inverse, to see if it contains all the specified *device coordinate points*. (If the view transform has no inverse, it is not considered.) This function returns the *index* of the view with the highest view transformation input priority that contains all of the points. If no view contains all the points, then the *view index* 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) are not transformed nor returned, so the number of points returned may be less than the number sent.

PEXlib allocates memory for the return value. Invoke **XFree** to deallocate the memory.

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 [§2.3.24, page 29].
wc_count	The number of world coordinate points.
wc_points	An array of world coordinate points [§2.7.1, page 33].
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 [§2.26.3, page 62].

Returns

Zero if unsuccessful; non-zero otherwise.

Description

Invoke PEXMapWCToDC to map the world coordinate points to device coordinate points using the specified *workstation* and *view index*.

This function transforms the points to device coordinates by passing them through the view transform associated with the view index. Points that are clipped (outside the viewport) are not returned, so the number of points returned may be less than the number sent.

PEXlib allocates memory for the return value. Invoke **XFree** to deallocate the memory.

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 [§2.3.24, page 29].
structure	The resource identifier of the structure [§2.3.19, page 29].
priority	The priority of the newly-posted structure.

Returns

None.

Description

Invoke **PEXPostStructure** to add the specified *structure* to the list of posted structures in the specified *workstation*.

Use the *priority* argument to indicate the priority of the new posted structure relative to the structures already posted. If you post multiple structures for display to the same display space location, then the higher priority structure is displayed. If two structures have the same priority, then PEXlib considers the last posted structure to be of higher priority.

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadPEXWorkstation	The specified workstation resource identifier is invalid.

See Also

PEXUnpostAllStructures, **PEXUnpostStructure**,

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 [§2.3.24, page 29].

Returns

None.

Description

Invoke **PEXRedrawAllStructures** to redraw all of the posted structures for the specified *workstation*.

First, if the workstation's display surface is **PEXNotEmpty**, then this function clears its drawable 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**, this function makes the requested values current and sets the update attributes 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**.

Errors

BadPEXWorkstation	The specified workstation resource identifier is invalid.

See Also

PEXExecuteDeferredActions, **PEXUpdateWorkstation**,

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 [§2.3.24, page 29].
count	The number of clip rectangles.
clip_rectangles	An array of device clip rectangles [§2.26.4, page 62].

Returns

None.

Description

Invoke **PEXRedrawClipRegion** to perform actions similar to **PEXRedrawAllStructures** except that this function does not modify nor update the workstation state attributes and rendering occurs only in the region defined by the specified clip list.

This function uses the color stored in entry zero of the color table to clear the region defined by the clip list. The clip list must consist of non-overlapping rectangles or the result is undefined. If a z-buffering HLHSR algorithm is used, then only the z-values corresponding to pixels in the clip region are affected. All of the posted structures for the workstation are redrawn, though they are clipped to the clip list. This function does not make pending changes current nor does it modify the visual state.

Errors

BadPEXWorkstation	The specified workstation resource identifier is invalid.

See Also

PEXExecuteDeferredActions, PEXRedrawAllStructures

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 [§2.3.24, page 29].
buffer_mode	The workstation buffering mode (**PEXSingleBuffer** or **PEXDoubleBuffer**).

Returns

None.

Description

Invoke **PEXSetWorkstationBufferMode** to set 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**, then this function sets the current buffer mode to the specified *buffer mode* and sets the buffer update to **PEXNotPending**. Otherwise, the buffer update is set to **PEXPending** and the current buffer mode is not changed.

PEXlib defines the following buffer modes: **PEXSingleBuffer** or **PEXDoubleBuffer**. If the buffer mode is PEXDoubleBuffer and the implementation cannot allocate the second image buffer, then the implementation issues a *BadAlloc* error.

Errors

BadAlloc	The implementation 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

PEXGetImpDepConstants, PEXSetWorkstationDisplayUpdateMode

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 [§2.3.24, page 29].
update_mode	The workstation display update mode (**PEXVisualizeEach**, **PEXVisualizeEasy**, **PEXVisualizeNone**, **PEXSimulateSome** or **PEXVisualizeWhenever**).

Returns

None.

Description

Invoke **PEXSetWorkstationDisplayUpdateMode** to set the *display update* attribute of the specified *workstation*.

The *display update* attribute defines how changes to the display surface are visualized. To inquire the supported values for *display update mode*, invoke the **PEXGetEnumTypeInfo** function.

If double-buffering is enabled, then the *display update mode* affects which buffer is rendered into during traversal. If the *display update mode* is **PEXVisualizeEach**, **PEXVisualizeWhenever** or **PEXVisualizeNone**, then 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**, then output primitives are always rendered into the front (displayed) buffer.

Errors

BadPEXWorkstation	The specified workstation resource identifier is invalid.
BadValue	The specified display update mode is invalid.

See Also

PEXGetEnumTypeInfo, PEXSetWorkstationBufferMode

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 [§2.3.24, page 29].
hlhsr_mode	The workstation HLHSR mode (**PEXHLHSROff**, **PEXHLHSRZBuffer**, **PEXHLHSRPainters**, **PEXHLHSRScanline**, **PEXHLHSRHiddenLineOnly** or, **PEXHLHSRZBufferID**).

Returns

None.

Description

Invoke **PEXSetWorkstationHLHSRMode** to set 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**, then this function sets the current HLHSR mode to the specified *HLHSR mode* and sets the HLHSR update to **PEXNotPending**; otherwise, it sets the HLHSR update to **PEXPending** and does not change the current HLHSR mode.

Errors

BadAlloc	The implementation 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 [§2.3.24, page 29].
viewport	The workstation viewport [§2.26.8, page 62].

Returns

None.

Description

Invoke **PEXSetWorkstationViewport** to set the requested workstation *viewport* of the specified *workstation*.

If the dynamic modification for workstation viewport is **PEXIMM**, then this function sets the current workstation viewport to the specified *viewport* and sets the workstation update to **PEXNotPending**; otherwise, it sets the workstation update to **PEXPending** and does not change the current workstation viewport.

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 [§2.3.24, page 29].
index1	The first view table entry index.
index2	The second view table entry index.
priority	A relative priority of *index1* with respect to *index2* (**PEXHigher** or **PEXLower**).

Returns

None.

Description

Invoke **PEXSetWorkstationViewPriority** to set 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.

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 [§2.3.24, page 29].
view_index	The view index.
values	A pointer to the view representation values [§2.16.29, page 45].

Returns

None.

Description

Invoke **PEXSetWorkstationViewRep** to set the specified view table entry of the requested view table in the specified *workstation*.

If the dynamic modification for the view table is **PEXIMM**, then this function sets the current view table entry to the specified view representation and sets the view update to **PEXNotPending**; otherwise, it sets the view update to **PEXPending** and does not change the current view table.

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 [§2.3.24, page 29].
workstation_window	The workstation window [§2.26.7, page 62].

Returns

None.

Description

Invoke **PEXSetWorkstationWindow** to set the requested NPC subvolume of the specified workstation.

If the dynamic modification for the workstation window is **PEXIMM**, then this function sets the current workstation window to the specified NPC subvolume and sets the workstation update to **PEXNotPending**; otherwise, it sets the workstation update to **PEXPending** and does not change the current workstation window.

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 [§2.3.24, page 29].

Returns

None.

Description

Invoke **PEXUnpostAllStructures** to remove all structures from the post structure list of the specified *workstation*.

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 [§2.3.24, page 29].
structure	The resource identifier of the structure [§2.3.19, page 29].

Returns

None.

Description

Invoke **PEXUnpostStructure** to remove the specified structure from the posted structure list of the specified workstation.

If the structure is not found in the list, then the implementation issues an error.

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 [§2.3.24, page 29].

Returns

None.

Description

Invoke **PEXUpdateWorkstation** to perform actions identical to **PEXRedrawAllStructures** on the specified *workstation* if the workstation's visual state is currently set to **PEXDeferred** or **PEXSimulated**.

Errors

BadPEXWorkstation	The specified workstation resource identifier is invalid.

See Also

PEXRedrawAllStructures

PEXSetPWAttributeMask - Setup Workstation Attributes Value Mask

Synopsis

void **PEXSetPWAttributeMask**(unsigned long **mask*, unsigned long *attr*)

Arguments

mask	The address of the value mask - an array of two unsigned long.
attr	A single workstation attribute bitmask constant.

Description

Use the **PEXSetPWAttributeMask** utility function as an aid in setting up the bitmask for the workstation attributes.

Use the following constants:

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

This macro does multiple evaluations of the value for *attr*.

Errors

None.

See Also

PEXGetWorkstationAttributes

PEXSetPWAttributeMaskAll - Set All Workstation Attributes in Value Mask

Synopsis

void **PEXSetPWAttributeMaskAll**(unsigned long **mask*)

Arguments

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

Description

Use the **PEXSetPWAttributeMaskAll** utility function as an aid in setting up the bitmask for the workstation attributes. This function sets all valid bits in the *mask*.

Errors

None.

See Also

PEXGetWorkstationAttributes

Chapter 20

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:

> **pick device** - a part of the PHIGS workstation resource, but there are functions that support the inquiring and setting of the pick device attributes (documented here).
>
> **pick measure** - a separate resource that works with the PHIGS workstation resource to perform the actual pick operation.

20.1. 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. You can set or inquire this list of values 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, then the contents of the name set remain, because the name set is still being referenced by the pick device descriptor. However, when you inquire a pick device descriptor, the value **PEXAlreadyFreed** is returned for the name set identifier, because it no longer has a valid resource identifier by which it can be referenced.

PEXlib defines the following pick device descriptor attributes (provided here in order):

> **pick status** - contains the initial pick status that will be bound to a pick measure resource.
>
> **pick path** - contains the initial pick path that will be bound to a pick measure resource.
>
> **pick path order** - specifies the order in which elements of the picked path are to be written into the pick measure resource. PEXlib defines the following orders:
>
> > **PEXTopFirst** - elements of the pick measure's picked path attribute are listed in the order they would have been encountered during a traversal.
> >
> > **PEXBottomFirst** - elements of the pick measure's picked path attribute are listed in the order opposite from the order they would have been encountered during a traversal.
>
> **pick inclusion** - specifies the resource identifier of the name set that is to be used as the pick inclusion filter during pick operations.
>
> **pick exclusion** - specifies the resource identifier of the name set that is to be used as the pick exclusion filter during pick operations.
>
> **pick data record** - contains a pick-device-dependent data record used to initialize a pick measure resource when it is created.
>
> **prompt and echo type** - contains a value that specifies the prompt and echo type to be used during pick operations. To inquire the supported types, invoke the **PEXGetEnumTypeInfo** function.

echo volume - specifies where prompting and echoing is to occur. By default, the echo volume assumes the size of the drawable bound to the workstation resource at the time it was created.

echo switch - specifies the initial echo state for a pick measure.

The pick device functions include:

“PEXChangePickDevice - Change Pick Device Attributes” on page 701 .
“PEXFreePDAttributes - Free Storage Returned by PEXGetPickDevice” on page 703.
“PEXGetPickDevice - Get Pick Device Attributes” on page 706.

20.2. Workstation Pick Measures

You must create a pick measure resource to perform a pick operation using a PHIGS workstation. When you create a pick measure resource, a pick device type is specified which provides the arguments for the pick operation. The pick measure accepts input in the form of input records. Input records are defined for each type of pick device. When the pick measure passes a valid input record, then its attributes are updated. Operations on a pick measure are potentially lengthy because to get the pick results, a great number of structure elements may have to be processed.

PEXlib defines the following pick measure resource attributes:

status - contains the result of the last update operation that was performed. It is set to **PEXPick** if a primitive was successfully picked, and set to **PEXNoPick** if the pick was unsuccessful.

pick path - contains the path of the structure element that was picked as a result of the last update operation.

The pick measure functions include:

“PEXCreatePickMeasure - Create Pick Measure” on page 702.
“PEXFreePickMeasure - Free Pick Measure” on page 704.
“PEXFreePMAttributes - Free Storage Returned by PEXGetPickMeasure” on page 705.
“PEXGetPickMeasure - Get Pick Measure Attributes” on page 707.
“PEXUpdatePickMeasure - Update Pick Measure” on page 708.

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 [§2.3.24, page 29].
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 [§2.23.3, page 52].

Returns

None.

Description

Invoke **PEXChangePickDevice** to modify the attributes of a pick descriptor for the PHIGS *workstation* resource specified.

This function modifies the descriptor which is the currently-defined descriptor for the pick device of the *pick device type* specified. Inquire the supported *pick device types* by invoking the **PEXGetEnumTypeInfo** function.

Use the *value mask* to indicate 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

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

PEXGetEnumTypeInfo, **PEXGetPickDevice**

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 [§2.3.24, page 29].
pick_device_type	The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).

Returns

The resource identifier of the newly-created pick measure [§2.3.15, page 28].

Description

Invoke **PEXCreatePickMeasure** to create a pick measure resource of the type specified.

This function initializes the pick measure with the values contained in the appropriate pick device descriptor stored in the specified *workstation*. Inquire the supported pick device types by invoking the **PEXGetEnumTypeInfo** function.

Errors

BadAlloc	The implementation 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**

PEXFreePDAttributes - Free Storage Returned by PEXGetPickDevice

Synopsis

void **PEXFreePDAttributes**(`PEXPDAttributes` **values*)

Arguments

values A pointer to the pick device attribute values [§2.23.3, page 52].

Returns

None.

Description

Invoke **PEXFreePDAttributes** to deallocate memory returned by **PEXGetPickDevice**.

Errors

None.

See Also

PEXGetPickDevice

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 [§2.3.15, page 28].

Returns

None.

Description

Invoke **PEXFreePickMeasure** to delete the *pick measure* resource and free the 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 [§2.23.4, page 52].

Returns

None.

Description

Invoke **PEXFreePMAttributes** to deallocate memory returned by **PEXGetPickMeasure**.

Errors

None.

See Also

PEXGetPickMeasure

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 [§2.3.24, page 29].
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 [§2.23.3, page 52]; a null pointer if unsuccessful.

Description

Invoke **PEXGetPickDevice** to get the attribute values of a pick descriptor for the PHIGS *workstation* resource specified.

This function returns a descriptor which is the currently-defined descriptor for the pick device of the *pick device type* specified. Inquire the supported *pick device types* by invoking the **PEXGetEnumTypeInfo** function.

Use the *value mask* to indicate 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. Invoke **PEXFreePDAttributes** to deallocate the memory.

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**

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 [§2.3.15, page 28].
value_mask	A mask indicating which attributes to return.

Returns

A pointer to the pick measure attribute values [§2.23.4, page 52]; a null pointer if unsuccessful.

Description

Invoke **PEXGetPickMeasure** to get the attribute values of the specified *pick measure*.

Use the *value mask* to indicate 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. Invoke **PEXFreePMAttributes** to deallocate the memory.

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 [§2.3.15, page 28].
pick_device_type	The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).
pick_record	A pointer to the pick data record [§2.25.8, page 60].

Returns

None.

Description

Invoke PEXUpdatePickMeasure to update the specified *pick measure* resource.

If the update causes a primitive to be picked, then this function sets the pick measure's pick status to **PEXPick** and sets the pick path attribute to the path of the picked primitive. If no primitive was picked, then this function sets the pick status to **PEXNoPick**.

Use the pick path 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. PEXlib defines the following values for *pick_device_type*:

PEXPickDeviceDCHitBox - point the input record to a data structure of type `PEXPDDCHitBox`.

PEXPickDeviceNPCHitVolume - point the input record to a data structure of type `PEXPDNPCHitVolume`.

Errors

BadPEXPath	The specified path is invalid.
BadPEXPickMeasure	The specified pick measure resource identifier is invalid.

See Also

PEXFreePickMeasure

Chapter 21

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 text fonts that you can scale and rotate. Because of the added capabilities of PEX fonts, PEX has defined its own PEX font 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 is displayed when an undefined or non-existent glyph is used. If the default glyph specifies a non-existent glyph, then nothing is 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 supports stroke precision, which implies the font is able to be scaled and rotated. The stroke font also enables the application to 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. PEX 5.2 implementations are required to have a XLFD font name for each available font. These implementations may also provide alternate names for some fonts, for backwards compatibility. Each implementation is required to support at least one font which supports the character set ISO 8859/1 (Latin1) and is of at least medium weight. The XLFD name for this font must match the following XLFD match pattern:

    -*-*-*-*-*-*-*-*-*-*-*-*-ISO8859-1

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 [§2.13.2, page 37].

Returns

None.

Description

Invoke **PEXFreeFontInfo** to deallocate memory returned by the **PEXListFontsWithInfo** and **PEXQueryFont** functions.

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

Invoke **PEXFreeFontNames** to deallocate memory returned by the **PEXListFonts** and **PEXListFontsWithInfo** functions.

Errors

None.

See Also

PEXListFonts, **PEXListFontsWithInfo**

PEXListFonts - List PEX Fonts (5.0)

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

Invoke **PEXListFonts** which is similar to X11 **XListFonts** except that this function only returns information about fonts that can be used by PEX.

The returned 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. 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. Invoke **PEXFreeFontNames** to deallocate the memory.

Errors

None.

See Also

PEXFreeFontNames, **PEXListFontsWithInfo**,

PEXListFontsWithInfo - List PEX Fonts With Information (5.0)

Synopsis

char ****PEXListFontsWithInfo**(Display **display*, char **pattern*, unsigned int *max_names*, unsigned long **count_return*, `PEXFontInfo` ***font_info*)

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) [§2.13.2, page 37].

Returns

An array of font names (null-terminated strings); a null pointer if unsuccessful or if no PEX fonts are supported.

Description

Invoke **PEXListFontsWithInfo** which is similar to X11 **XListFontsWithInfo** except that this function only returns information about fonts that can support the full range of PEX text attributes (i.e., those fonts that are "PEX usable").

The returned 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. 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 that which **PEXQueryFont** returns.

PEXlib allocates memory for the returned list of font names and font info. Invoke **PEXFreeFontNames** to deallocate the memory for the font names, as pointed to by the function's return value. Invoke **PEXFreeFontInfo** to deallocate the memory for the font information located by the pointer returned in *font_info*.

Errors

None.

See Also

PEXFreeFontInfo, PEXFreeFontNames, **PEXListFonts**, **PEXQueryFont**

PEXLoadFont - Load PEX Font (5.0)

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 [§2.3.7, page 28].

Description

Invoke **PEXLoadFont** to load the specified PEX *font* and return a `PEXFont` which identifies it. The implementation does not load multiple copies of the font, even if multiple requests to load it are received.

The *font name* should follow the X Logical Font Description (XLFD) naming conventions. For backwards compatibility, the font name can also be any string encoded in ISO Latin-1. Case is not significant in matching names. PEX fonts are not associated with a particular screen, and you can use them with any renderer or PHIGS workstation resource. You may use **PEXListFonts** or **PEXListFontsWithInfo** to determine which fonts are available and which support the full stroke precision.

If you specify a font that is not "PEX usable" then the implementation issues a *BadPEXFont* error.

Errors

BadAlloc	The implementation failed to allocate the resource.
BadPEXFont	The specified font name does not name a usable PEX font.

See Also

PEXListFonts, **PEXListFontsWithInfo, PEXUnloadFont**

PEXQueryEncodedTextExtents - Query Encoded Text Extents (5.0)

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** or **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** or **PEXHAlignRight**).
valign	The text vertical text alignment (**PEXVAlignNormal**, **PEXVAlignTop**, **PEXVAlignCap**, **PEXVAlignHalf**, **PEXVAlignBase** or **PEXVAlignBottom**).
count	The number of encoded text strings.
encoded_text	An array of encoded text strings [§2.13.5, page 38].

Returns

An array of text extents; a null pointer if unsuccessful [§2.13.8, page 38].

Description

Invoke **PEXQueryEncodedTextExtents** which is similar to **PEXQueryTextExtents**, except that this function specifies multiple encoded text strings.

With this function, 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` [§1.7.1, page 28])

The current font group is taken to be that entry of the Text Font lookup table associated with the *resource_id* selected by the *font_table_index*. 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 selects the third font in the font group currently being used. If a character set of any of the encoded text strings is not available in the current font group, then the default font group is used for the array of encoded text strings. 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 number of bytes per glyph in each encoded string. PEXlib defines three values for character set width: **PEXCSByte**, **PEXCSShort** and **PEXCSLong**. PEXlib provides the encoding state for your use but the PEX implementation does not interpret the encoding state.

All other aspects of this function are the same as **PEXQueryTextExtents**.

Errors

BadMatch	The specified resource identifier identifies a table of a type other than a text font table.
BadValue	A specified value for one or more 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 (5.0)

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 [§2.3.7, page 28].

Returns

A pointer to the font info structure; a null pointer if unsuccessful [§2.13.2, page 37].

Description

Invoke **PEXQueryFont** to get information about the specified PEX font.

The information returned is identical to that which **PEXListFontsWithInfo** returns.

PEXlib allocates memory for the returned list of font information. Invoke **PEXFreeFontInfo** to deallocate the memory.

Errors

BadPEXFont	The specified font resource identifier is invalid.

See Also

PEXFreeFontInfo

PEXQueryTextExtents - Query Text Extents (5.0)

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** or **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** or **PEXHAlignRight**).
valign	The text vertical text alignment (**PEXVAlignNormal**, **PEXVAlignTop**, **PEXVAlignCap**, **PEXVAlignHalf**, **PEXVAlignBase** or **PEXVAlignBottom**).
count	The number of text strings.
text	An array of text strings [§2.13.6, page 38].

Returns

An array of text extents; a null pointer if unsuccessful [§2.13.8, page 38].

Description

Invoke **PEXQueryTextExtents** to get the geometric extent and the concatenation point for each specified *text* string. The extent of a string is the smallest rectangle completely surrounding it.

The concatenation point is the position to place another string if the string is to be extended. The client may still need to apply a suitable modeling transformation to account for the character up vector.

If the *resource identifier* is a renderer or PHIGS workstation, then this function uses the text font table associated with the renderer or workstation. If the *resource identifier* is a text font lookup table, then this function uses it directly. The specified *font_table_index* provides the index of the entry that this function uses to get the font group. The first character set in the font group is used for all strings in *text*. If that font group does not define a font for character set 1, then the implementation substitutes the font for character set 1 of the default font group.

Stroke precision is assumed.

The extent and concatenation point of each string is computed independently of the other strings in the list.

If the selected font has no defined default glyph, then this function assumes undefined glyphs have all zero metrics.

The extent data is returned in memory allocated by PEXlib. Invoke **XFree** to deallocate the memory.

Errors

BadMatch	The specified resource identifier identifies a table of a type other than a

text font table.

BadValue A specified value for one or more text attributes is invalid, or the specified resource identifier does not identify a valid renderer, workstation or lookup table resource.

See Also

PEXQueryEncodedTextExtents

PEXUnloadFont - Unload PEX Font (5.0)

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 [§2.3.7, page 28].

Returns

None.

Description

Invoke **PEXUnloadFont** to delete the association between the resource identifier and the PEX font.

The PEX font itself is freed when no other resources references it.

Errors

BadPEXFont	The specified font resource identifier is invalid.

See Also

PEXLoadFont

Chapter 22

PEX Escapes

PEXEscape and **PEXEscapeWithReply** are functions which provide a standard interface to functionality not defined by PEX but available in a particular PEX implementation. **PEXSetEchoColor** is a convenience function for easy access to the one standard PEX escape.

The **PEXEscape** and **PEXEscapeWithReply** functions each require an escape ID and a data record containing any data needed by the escape. The entity defining the escape defines the ID and the encoding of the escapes's data record and reply data record, if needed. In general, the entity defining the escape provides C language header files containing defined constants for the IDs and structure type definitions for the input and reply data records. You can use these constants and definitions directly in the escape functions' arguments. The standard place to locate these header files is **<X11/PEX5/extensions>**.

Before attempting to use an escape, the client program should ensure that the implementation supports the escape by checking the list of supported escapes returned by the **PEXGetEnumeratedTypeInfo** function.

PEXEscape - PEX Escape (5.1)

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

Invoke **PEXEscape** to access implementation-specific functionality.

This request invokes the PEX-defined or vendor-specific escape specified by *escape_id*. The complete interface and behavior for each specified escape identifier should be specified by the individual PEX implementation.

PEXlib defines one standard PEX Escape:

> **PEXEscapeSetEchoColor** - sets 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. This escape is defined for compatibility with PEX 5.1. Starting with PEX 5.2, this function is implemented with the echo color renderer attribute.

Errors

Escape-dependent	See documentation provided with the individual PEX implementation.
BadValue	The specified escape identifier is unsupported or is the identifier of an escape that generates a reply.

See Also

PEXEscapeWithReply, **PEXSetEchoColor**

PEXEscapeWithReply - PEX Escape With Reply (5.1)

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

Invoke **PEXEscapeWithReply** to access implementation-specific functionality. This request is similar to **PEXEscape** except it has return data.

This request invokes the PEX-defined or vendor-specific escape specified by *escape_id*. The complete interface and behavior for each specified escape identifier should be specified by the individual PEX implementation.

The reply data is returned in memory allocated by PEXlib. Invoke **XFree** to deallocate the memory. The reply data consists of all the data in the protocol reply packet, starting with the 13^{th} byte of the protocol reply packet.

There are no standard PEX escapes with replies.

Errors

Escape-dependent	See documentation provided with the individual PEX implementation.
BadValue	The specified escape identifier is unsupported or is the identifier of an escape that does not generate a reply.

See Also

PEXEscape

PEXSetEchoColor - PEX Escape to set Echo Color for Specified Renderer (5.1)

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** or **PEXColorTypeRGB16**).
color	The echo color [§2.5.12, page 32].

Returns

None

Description

Invoke **PEXSetEchoColor** to conveniently access the **PEXEscapeSetEchoColor** escape in order to set the echo color for use by a renderer when echoing primitives.

This function is defined for compatibility with PEX 5.1 and PEXlib 5.1. Beginning with PEX 5.2 and PEXlib 5.2, you should instead use the standard echo color renderer attribute to control the renderer's echo color.

You can change a renderer's echo color at any time during rendering, so the echo color is fully dynamic. You cannot inquire the dynamics of the echo color (see **PEXGetRendererDynamics**). Lighting and shading for echoed primitives is implementation-dependent. You cannot inquire a renderer's echo color and the default value is implementation-dependent. Inquire the support of this escape by invoking the **PEXGetEnumTypeInfo** function.

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 23

Output Command Encode and Decode

Encode and decode output commands are provided for applications that need to generate protocol-formatted output commands for use in archives or client-side data storage. Likewise, your 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*.

For the definition of the `PEXOCData` data structure, please refer to the PEXlib.h file that accompanies your PEX implementation. The data structure is too large to reproduce here in its entirety.

Handling exception OC's defined after 5.1

Using the `PEXOCData` structure with the encode and decode functions is straightforward for all the output commands defined in PEX 5.1 and PEX 5.2 except for **SetLocalTransformWithHints**. Except for this single OC, you simply use an array of `PEXOCData`, one element for each OC, and fill in or read the OC data in each `PEXOCData` element, using one of the structs in the `PEXOCData` union, selected by the OC type.

Here is an abbreviated version of the `PEXOCData` data structure:

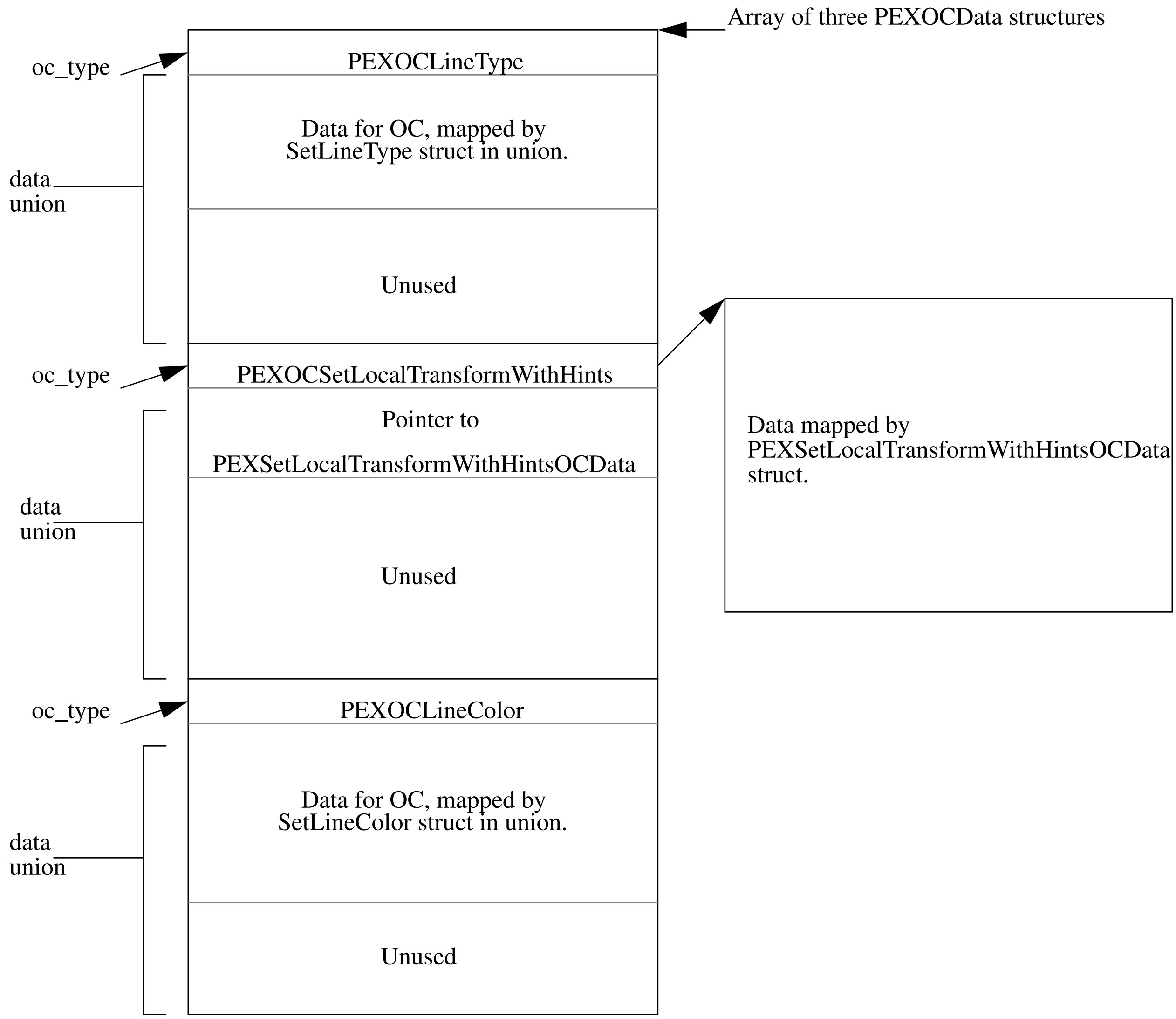
```
typedef struct {
      unsigned short oc_type;
      union {
            struct {
                  unsigned long count;
                  PEXName *names;
            } AddToNameSet;
            ...
            < additional structs for 5.1 and 5.2 OC's >
            ...
            struct {
                  PEXSetLocalTransformWithHintsOCData *ocptr;
            } SetLocalTransformWithHints;
            ...
            < other structs for 5.2 OC's >
      } data;
} PEXOCData;

typedef struct {
      ...
} PEXSetLocalTransformWithHintsOCData;
```

The **SetLocalTransformWithHints** OC is the only 5.2 OC larger in size than the largest 5.1 OC, and would cause the size of the `PEXOCData` structure to increase in size if a new structure was simply added to the `PEXOCData` structure. This would break binary compatibility between applications compiled with PEXlib 5.1 header files, but linked (dynamically) with PEXlib 5.2 library implementations.

The solution specified for this OC is to use a pointer in the `SetLocalTransformWithHints` structure to point to another disjoint area of storage that contains the OC data for this OC. This storage area is mapped with the `PEXSetLocalTransformWithHintsOCData` structure.

Here is a diagram of the data storage layout for an array of PEXOCData structures that contains a

SetLocalTransformWithHints output command:



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** or **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

Invoke **PEXCountOCs** to get the number of output commands in the encoded list of output commands.

This function has no visible effect.

If your implementation does not support the specified floating-point format, then this function returns a count of zero.

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** or **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 [§2.18.2, page 46]; a null pointer if unsuccessful or if zero output commands are specified.

Description

Invoke **PEXDecodeOCs** to convert an encoded stream of output commands into an array of `PEXOCData` data structures.

This function has no visible effect.

The data typically passed as parameters to output attribute or primitive functions is returned in memory allocated by PEXlib. Invoke **PEXFreeOCData** to deallocate the memory.

If your implementation does not support the specified floating-point format, then this function returns a null pointer.

The **PEXDecodeOCs** function builds `PEXOCData` data structures based on the protocol output command type. You should decode the rest of the output command based on its type and not the PEXlib function that originally created the output command. Many output commands are generated by more than one PEXlib function.

Invoke **PEXCountOCs** 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** or **PEXDEC_D_Floating**).
oc_count	The number of output commands to be encoded.
oc_data	An array of the output command data [§2.18.2, page 46].
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 are specified.

Description

Invoke **PEXEncodeOCs** to assemble an encoded PEX protocol representation of output commands described by an array of `PEXOCData` data structures.

This function has no visible effect.

The data typically passed as parameters to output attribute or primitive functions is passed. The encoded data is returned in memory allocated by PEXlib. Invoke **XFree** to deallocate the memory.

If your implementation does not support the specified floating-point format, then this function returns a null pointer.

You must specify any text or annotation text primitives 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 [§2.18.2, page 46].

Returns

None.

Description

Invoke **PEXFreeOCData** to deallocate memory allocated by PEXlib to hold decoded output command data.

Errors

None.

See Also

PEXDecodeOCs

Chapter 24

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. You can invoke **PEXCopyBytesToOC** to have PEXlib copy individual pieces which are in contiguous memory. Invoke **PEXGetOCAddr** 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 implementation and do any necessary conversion on the data. For example, if the PEX implementation does not support the floating-point format native to the application's machine, then the application must convert any native float values into a format supported by the PEX implementation.

For complete information on protocol format, see the *PEX Protocol Specification*.

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

Invoke **PEXCopyBytesToOC** to copy the specified number of bytes of *data* to the transport buffer.

It is recommended that the number of bytes be a multiple of four because 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.

You must first invoke **PEXStartOCs**.

Errors

None.

See Also

PEXFinishOCs, **PEXGetOCAddr**, **PEXStartOCs**

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

Invoke **PEXFinishOCs** to complete the sending of encoded output commands.

This function unlocks the display connection.

Errors

None.

See Also

PEXCopyBytesToOC, **PEXGetOCAddr, PEXStartOCs**

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

Invoke **PEXGetOCAddr** to get a memory address to the specified number of bytes in the transport buffer where the application can write data. This function flushes data currently in the transport buffer, if needed to satisfy the requested *length*.

This function returns a pointer that is valid only until you invoke the next **PEXGetOCAddr** or **PEXCopyBytesToOC**.

An attempt to request more bytes than returned by **PEXGetOCAddrMaxSize**, results in an unsuccessful return value (NULL pointer).

You must first invoke **PEXStartOCs**.

Do *not* attempt to deallocate or free memory at the address returned by this function.

Errors

None.

See Also

PEXCopyBytesToOC, **PEXFinishOCs**, **PEXGetOCAddrMaxSize**, **PEXStartOCs**

PEXGetOCAddrMaxSize - Obtain the Maximum Size for PEXGetOCAddr

Synopsis

int **PEXGetOCAddrMaxSize**(Display **display*)

Arguments

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

Returns

The size, in bytes, of the maximum size area in the transport buffer that can be obtained by **PEXGetOCAddr**.

Description

Use the **PEXGetOCAddrMaxSize** function to evaluate the maximum size for the *length* argument of **PEXGetOCAddr**. This value is an implementation-dependent constant and does not reflect the current utilization of the transport buffer.

Errors

None.

See Also

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** or **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

Invoke **PEXSendOCs** to send encoded output commands to the specified PEX implementation.

Sending output commands to a structure which has an editing mode of **PEXStructureReplace**, is not really useful. The behavior is unpredictable unless a request type of **PEXOCStoreSingle** is used. If you specify a request type of **PEXOCStoreSingle**, then each output command simply replaces the previous one sent. You should ensure that the structure's editing mode is **PEXStructureInsert**, when sending multiple output commands. If it is intended to replace multiple elements, then 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** or **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

Invoke **PEXStartOCs** to lock the display.

You can invoke only **PEXCopyBytesToOC** or **PEXGetOCAddr** between pairs of **PEXStartOCs** and **PEXFinishOCs** functions. Do not invoke anything else that may lock the display; this results in a deadlock.

The first output command is guaranteed to start on a four-byte boundary. You can copy output command data into the transport buffer by calling **PEXCopyBytesToOC**. Output command data can be written directly by the application by calling **PEXGetOCAddr** to get a pointer to memory in the transport buffer.

You must invoke **PEXFinishOCs** 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), then this function returns unsuccessfully. If this occurs, and the number of output commands was greater than one, 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, then this function returns 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

PEXCopyBytesToOC, **PEXFinishOCs**, **PEXGetOCAddr**

Chapter 25

PEXlib Utilities

This chapter specifies the PEXlib utility functions. Use these functions to perform the tasks that are listed below.

Note: These utilities require that you include the math library *libm* when linking your application.

The utilities are organized into the following areas of functionality:

- Modeling Transformation Utilities
- Viewing Transformation Utilities
- Miscellaneous Transformation Utilities
- Utilities for Computing Geometric Normals
- Utilities for Computing Shape Hints

25.1. 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:

PEXBuildTransform
PEXBuildTransform2D
PEXMatrixMult
PEXMatrixMult2D
PEXRotate
PEXRotate2D
PEXRotateGeneral
PEXScale
PEXScale2D
PEXTranslate
PEXTranslate2D

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 [§2.7.1, page 33].
trans_vector	Translation vector [§2.31.1, page 67].
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* [§2.31.1, page 67].
matrix_return	Matrix into which result is stored [§2.3.10, page 28].

Returns

None.

Description

Invoke **PEXBuildTransform** to build 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 [§2.7.2, page 33].
trans_vector	Translation vector [§2.31.2, page 67].
angle_z	Angle of rotation about *z* axis, in radians.
scale_vector	Vector of scale factors for *x* and *y* axes [§2.31.2, page 67].
matrix_return	Matrix in which result is stored [§2.3.9, page 28].

Returns

None.

Description

Invoke **PEXBuildTransform2D** to build a 3x3 transformation matrix that scales by the values in the *scale vector* about the *fixed point*, rotates about the *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

PEXMatrixMult

Synopsis

void **PEXMatrixMult** (`PEXMatrix` *matrix1*, `PEXMatrix` *matrix2*, `PEXMatrix` *matrix_return*)

Arguments

matrix1	First matrix to be multiplied [§2.3.10, page 28].
matrix2	Second matrix to be multiplied [§2.3.10, page 28].
matrix_return	Matrix into which result is stored [§2.3.10, page 28].

Returns

None.

Description

Invoke **PEXMatrixMult** to perform a matrix multiplication: *matrix_return* = *matrix1* x *matrix2*.

If the return matrix is the same as one of the input matrices, then this utility overwrites 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 [§2.3.9, page 28].
matrix2	Second matrix to be multiplied [§2.3.9, page 28].
matrix_return	Matrix into which result is stored [§2.3.9, page 28].

Returns

None.

Description

Invoke **PEXMatrixMult2D** to perform a 3x3 matrix multiplication: *matrix_return* = *matrix1* x *matrix2*.

If the return matrix is the same as one of the input matrices, then this utility overwrites the input values with the multiplied values.

Errors

None.

See Also

PEXMatrixMult

PEXRotate

Synopsis

int **PEXRotate**(int *axis*, double *angle*, `PEXMatrix` *matrix_return*)

Arguments

axis	The axis about which you want the rotation to occur (**PEXXAxis**, **PEXYAxis** or **PEXZAxis**).
angle	Angle of the rotation in radians.
matrix_return	Matrix into which rotation matrix is stored [§2.3.10, page 28].

Returns

Zero if successful; otherwise, one of the following:

PEXBadAxis	Invalid axis value specified

Description

Invoke **PEXRotate** to create a rotation matrix about the specified axis.

The resulting matrix rotates by the angle specified in radians about the origin.

This utility returns unsuccessfully if the specified *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 [§2.3.9, page 28].

Returns

None.

Description

Invoke **PEXRotate2D** to create a rotation matrix for the specified *angle*.

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 [§2.7.1, page 33].
point2	Endpoint of line segment defining rotation axis [§2.7.1, page 33].
angle	Angle of the rotation in radians.
matrix_return	Matrix into which rotation matrix is stored [§2.3.10, page 28].

Returns

Zero if successful; otherwise, one of the following:

PEXBadAxis	The endpoints are coincident

Description

Invoke **PEXRotateGeneral** to format 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.

This utility 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 [§2.31.1, page 67].
matrix_return	Matrix into which rotation matrix is stored [§2.3.10, page 28].

Returns

None.

Description

Invoke **PEXScale** to create a scale matrix according to the specified *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 [§2.31.2, page 67].
matrix_return	Matrix into which rotation matrix is stored [§2.3.9, page 28].

Returns

None.

Description

Invoke **PEXScale2D** to create a 3x3 scale matrix according to the specified *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 [§2.31.1, page 67].
matrix_return	Matrix into which rotation matrix is stored [§2.3.10, page 28].

Returns

None.

Description

Invoke **PEXTranslate** to create a translation matrix that translates objects by the specified *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 [§2.31.2, page 67].
matrix_return	Matrix into which rotation matrix is stored [§2.3.9, page 28].

Returns

None.

Description

Invoke **PEXTranslate2D** to create a 3x3 translation matrix that translates objects by the specified *translation vector.*

Errors

None.

See Also

PEXTranslate

25.2. 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:

PEXViewMappingMatrix
PEXViewMappingMatrix2D
PEXViewOrientationMatrix
PEXViewOrientationMatrix2D

Those for the simple viewing model are:

PEXLookAtViewMatrix
PEXOrthoProjMatrix
PEXPerspProjMatrix
PEXPolarViewMatrix

You can pass the matrices returned by these functions as viewing transforms to PEXlib.

PEXLookAtViewMatrix

Synopsis

int **PEXLookAtViewMatrix**(`PEXCoord` **from*, `PEXCoord` **to*, `PEXVector` **up*, `PEXMatrix` *matrix_return*)

Arguments

from	Viewing position, in world coordinates [§2.7.1, page 33].
to	Look at position, in world coordinates [§2.7.1, page 33].
up	Vector representing the up direction, in world coordinates [§2.31.1, page 67].
matrix_return	Matrix in which result is stored [§2.3.10, page 28].

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

Invoke **PEXLookAtViewMatrix** to create a view orientation transform that defines the viewing direction and orientation. This utility 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 arguments 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, **PEXViewMappingMatrix**, **PEXViewOrientationMatrix**,

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 [§2.3.10, page 28].

Returns

Zero if successful; otherwise, one of the following:

PEXBadLimits	The viewing box depth, width, or height is zero.

Description

Invoke **PEXOrthoProjMatrix** to create 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, **PEXViewMappingMatrix**, **PEXViewOrientationMatrix**,**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 from the VRC origin.
aspect	The aspect ratio (width/height) of the perspective viewing frustum.
near	The distance to the near clipping plane from the VRC origin.
far	The distance to the far clipping plane from the VRC origin.
matrix_return	Matrix in which result is stored [§2.3.10, page 28].

Returns

Zero if successful; otherwise, one of the following:

PEXBadLimits near <= far, fovy = 0, aspect = 0, or distance <= near

Description

Invoke **PEXPerpProjMatrix** to create 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 invokes **PEXLookAtViewMatrix** to calculate the view orientation matrix, then the distance is typically the distance between the *from* and *to* points specified to that utility. If the application program invokes **PEXPolarViewMatrix** to calculate the view orientation matrix, then 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:

$$\tan(\text{fovy}/2) = ((\text{height}/2) / \text{eye\_distance})$$

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, **PEXOrthoProjMatrix**, **PEXViewMappingMatrix**, **PEXViewOrientationMatrix**,

PEXPolarViewMatrix

Synopsis

int **PEXPolarViewMatrix**(`PEXCoord` **from*, double *distance*, double *azimuth*, double *altitude*, double *twist*, `PEXMatrix` *matrix_return*)

Arguments

from	The viewing position [§2.7.1, page 33].
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 +*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 [§2.3.10, page 28].

Returns

Zero if successful; otherwise, one of the following:

PEXBadDistance	The distance specified is zero.

Description

Invoke **PEXPolarViewMatrix** to create a polar view orientation matrix. This utility is similar to **PEXLookAtViewMatrix**, except you specify the viewing arguments 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, **PEXViewMappingMatrix, PEXViewOrientationMatrix**,

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 two 2D VRC locations which mark a rectangle in the view plane [§2.7.2, page 33].
viewport	NPC viewport into which the frame gets mapped [§2.26.7, page 62].
perspective	Flag to indicate whether a perspective view is desired, a value of **True** requests that perspective be applied.
prp	Projection reference point [§2.7.1, page 33].
view_plane	VRC position of view plane with respect to the VRP.
back_plane	VRC position of the back plane with respect to the VRP.
front_plane	VRC position of the front plane with respect to the VRP.
matrix_return	Matrix into which result is stored [§2.3.10, page 28].

Returns

Zero if successful; otherwise, one of the following:

PEXBadLimits	The viewing box depth, width or height is zero.
PEXBadViewport	xmin >= xmax, or ymin >= ymax, or zmin > zmax
PEXBadPlanes	View plane coordinates invalid.
PEXBadPRP	Projection reference point invalid.

Description

Invoke **PEXViewMappingMatrix** to create 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 **PEXViewOrientationMatrix** (page 761)), The *y* axis is fixed by VUP (view up vector, see **PEXViewOrientationMatrix**), 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 is 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:

$xmin < xmax$, $ymin < ymax$, $zmin <= zmax$

Errors

None.

See Also

PEXViewMappingMatrix2D, **PEXViewOrientationMatrix**, **PEXViewOrientationMatrix2D**

PEXViewMappingMatrix2D

Synopsis

int **PEXViewMappingMatrix2D**(`PEXCoord2D` **frame*, `PEXCoord2D` **viewport*, `PEXMatrix3x3` *matrix_return*)

Arguments

frame	Array of two 2D VRC locations which mark a rectangle to be mapped into the NPC subvolume [§2.7.2, page 33].
viewport	Array of two 2D NPC coordinates specifying lower left and upper right of the viewport [§2.7.2, page 33].
matrix_return	Matrix in which result is stored [§2.3.9, page 28].

Returns

Zero if successful; otherwise, one of the following:

PEXBadLimits	The viewing box width or height is zero.
PEXBadViewport	xmin >= xmax, or ymin >= ymax

Description

Invoke **PEXViewMappingMatrix2D** as a two-dimensional shorthand version of **PEXViewMappingMatrix** with the following arguments/values 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* and *y* limits must be as follows:

xmin < *xmax*, *ymin* < *ymax*

Errors

None.

See Also

PEXViewMappingMatrix, **PEXViewOrientationMatrix**, **PEXViewOrientationMatrix2D**

PEXViewOrientationMatrix

Synopsis

int **PEXViewOrientationMatrix**(`PEXCoord` **vrp*, `PEXVector` **vpn*, `PEXVector` **vup*, `PEXMatrix` *matrix_return*)

Arguments

vrp	View reference point [§2.7.1, page 33].
vpn	View plane normal [§2.31.1, page 67].
vup	View up vector [§2.31.1, page 67].
matrix_return	Matrix into which result is stored [§2.3.10, page 28].

Returns

Zero if successful; otherwise, one of the following:

PEXBadVector	Either *vpn* or *vup* is zero length.
PEXBadVectors	*vup* is parallel to *vpn*

Description

Invoke **PEXViewOrientationMatrix** to create 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 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

PEXViewMappingMatrix, **PEXViewMappingMatrix2D**, **PEXViewOrientationMatrix2D**

PEXViewOrientationMatrix2D

Synopsis

`int` **PEXViewOrientationMatrix2D** (`PEXCoord2D` **vrp*, `PEXVector2D` **vup*, `PEXMatrix3x3` *matrix_return*)

Arguments

vrp	View reference point [§2.7.2, page 33].
vup	View up vector [§2.7.2, page 33].
matrix_return	Matrix in which result is stored [§2.3.9, page 28].

Returns

Zero if successful; otherwise, one of the following:

PEXBadVector	*vup* is zero length.

Description

Invoke **PEXViewOrientationMatrix2D** to create 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 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 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 is 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

PEXViewMappingMatrix, **PEXViewMappingMatrix2D**, **PEXViewOrientationMatrix**

25.3. Miscellaneous Transformation Utilities

This set of functions computes various useful transforms and applies transforms to points and vectors. Included in this set are:

PEXIdentityMatrix
PEXIdentityMatrix2D
PEXInvertMatrix
PEXInvertMatrix2D
PEXMapXCToNPC
PEXMapXCToNPC2D
PEXNormalizeVectors
PEXNormalizeVectors2D
PEXNPCToXCTransform
PEXNPCToXCTransform2D
PEXTransformPoints
PEXTransformPoints2D
PEXTransformPoints4D
PEXTransformPoints2DH
PEXTransformVectors
PEXTransformVectors2D
PEXXCToNPCTransform
PEXXCToNPCTransform2D

Use **PEXMapXCToNPC** or **PEXMapXCToNPC2D** to find the PEX view containing a drawable coordinate (XC) point. **PEXNPCToXCTransform** and **PEXNPCToXCTransform2D** compute the matrix representing the transformation from the NPC subvolume through the DC viewport and on to drawable coordinates. **PEXXCToNPCTransform** and **PEXXCToNPCTransform2D** compute the inverse of this transformation. You can use these utilities 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 utilities, you can transform a list of modeling coordinate points 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 A

    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.

PEXIdentityMatrix

Synopsis

void **PEXIdentityMatrix**(`PEXMatrix` *transform_return*)

Arguments

transform_return The returned identity matrix [§2.3.10, page 28].

Description

Invoke **PEXIdentityMatrix** to get an identity matrix.

Errors

None.

See Also

PEXIdentityMatrix2D

PEXIdentityMatrix2D

Synopsis

void **PEXIdentityMatrix2D**(`PEXMatrix3x3` *transform_return*)

Arguments

transform_return The returned identity matrix [§2.3.9, page 28]

Description

Invoke **PEXIdentityMatrix2D** to get a two-dimensional identity matrix.

Errors

None.

See Also

PEXIdentityMatrix

PEXInvertMatrix

Synopsis

int **PEXInvertMatrix**(`PEXMatrix` *transform*, `PEXMatrix` *transform_return*)

Arguments

transform	The transformation matrix to invert [§2.3.10, page 28].
transform_return	The inverse transformation [§2.3.10, page 28].

Returns

Zero if successful; non-zero if unsuccessful.

Description

Invoke **PEXInvertMatrix** to compute the inverse of a transformation matrix.

If the matrix is non-invertible, then this function returns a status of unsuccessful.

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 [§2.3.9, page 28].
transform_return	The inverse transformation [§2.3.9, page 28].

Returns

Zero if successful; non-zero if unsuccessful.

Description

Invoke **PEXInvertMatrix2D** to compute the inverse of a two-dimensional transformation matrix.

If the matrix is non-invertible, then this utility returns a status of unsuccessful.

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

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).
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 [§2.26.3, page 62].
npc_sub_volume	A pointer to an NPC subvolume, typically that of a renderer resource [§2.26.7, page 62]
view_count	The number of views to search
views	The view entries to search for inclusion of the transformed points [§2.16.29, page 45]
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	A pointer to an array in which to store the returned transformed points [§2.7.1, page 33].

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

Invoke **PEXMapXCToNPC** to map a list of drawable coordinates (XC) to NPC and search 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_return*.

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, then 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 [§2.25.2, page 59].
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 [§2.25.2, page 59].
npc_sub_volume	A pointer to an NPC subvolume, typically that of a renderer resource [§2.26.7, page 62].
view_count	The number of views to search.
views	The view entries to search for inclusion of the transformed points [§2.16.29, page 45].
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 [§2.7.2, page 33].

Returns

Zero if successful; otherwise, one of the following:

PEXBadViewport	xmin >= xmax, or ymin >= ymax
PEXBadSubVolume	xmin >= xmax, or ymin >= ymax, or zmin > zmax

Description

Invoke **PEXMapXCToNPC2D** to map a list of drawable coordinates (XC) to NPC and search a specified list of view entries to determine the view containing the computed NPC points.

The XC points are first transformed to two-dimensional DC, using the specified *window height*, then transformed to two-dimensional 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, then 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**

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 [§2.31.1, page 67].
vectors_return	A pointer to an array in which to store the normalized vectors [§2.31.1, page 67].

Returns

Zero if successful; otherwise, one of the following:

PEXBadVector	One of the vectors has zero magnitude.

Description

Invoke **PEXNormalizeVectors** to normalize each vector in the specified list of three-dimensional vectors.

This utility returns an error if any vector in the list has a magnitude of zero. However, this utility still normalizes all of the non-zero vectors in the list.

If the return array is the same as the input array, then this utility overwrites the input values with the normalized values.

Errors

None.

See Also

PEXNormalizeVectors2D, **PEXTransformVectors**

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 [§2.31.2, page 67].
vectors_return	A pointer to an array in which to store the normalized vectors [§2.31.2, page 67].

Returns

Zero if successful; otherwise, one of the following:

PEXBadVector	One of the vectors has zero magnitude.

Description

Invoke **PEXNormalizeVectors2D** to normalize each vector in the specified list of two-dimensional vectors.

This utility returns an error if any vector in the list has a magnitude of zero. However, this utility still normalizes all of the non-zero vectors in the list.

If the return array is the same as the input array, then this utility overwrites the input values with the normalized values.

Errors

None.

See Also

PEXNormalizeVectors, **PEXTransformVectors2D**

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 [§2.26.7, page 62].
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 [§2.26.3, page 62].
window_height	The height of the drawable.
transform_return	The returned transformation [§2.3.9, page 28].

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

Invoke **PEXNPCToXCTransform** to compute 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 [§2.26.7, page 62].
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 [§2.25.2, page 59].
window_height	The height of the drawable.
transform_return	The returned transformation [§2.3.9, page 28].

Returns

Zero if successful; otherwise, one of the following:

PEXBadViewport	xmin >= xmax, or ymin >= ymax
PEXBadSubVolume	xmin >= xmax, or ymin >= ymax, or zmin > zmax

Description

Invoke **PEXNPCToXCTransform2D** to compute the two-dimensional transformation matrix to map a two-dimensional 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 transforms 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

PEXTransformPoints

Synopsis

int **PEXTransformPoints**(`PEXMatrix` *transform*, int *count*, `PEXCoord` **points*, `PEXCoord` **points_return*)

Arguments

transform	The transformation matrix to apply to the points [§2.3.10, page 28].
count	The number of points to transform.
points	A pointer to an array of 3D points to transform [§2.7.1, page 33].
points_return	A pointer to an array in which to store the transformed points [§2.7.1, page 33].

Returns

Zero if successful; otherwise, one of the following:

PEXBadHomoCoord	One or more of the transformed points has a homogeneous coordinate of 0.

Description

Invoke **PEXTransformPoints** to apply the specified homogeneous *transformation matrix* to the list of *points*.

When applying the transformation, the points are first converted to homogeneous points by assigning them a homogeneous coordinate of 1. The transformation is then applied:

$$P' = T \mathrm{x} P$$

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, then all points other than those with a homogeneous coordinate of zero are transformed and returned.

If the return array is the same as the input array, then this utility overwrites the input values with the transformed values.

Errors

None.

See Also

PEXTransformPoints2D, **PEXTransformPoints2DH, PEXTransformPoints4D**

PEXTransformPoints2D

Synopsis

int **PEXTransformPoints2D**(`PEXMatrix3x3` *transform*, int *count*, `PEXCoord2D` **points*, `PEXCoord2D` **points_return*)

Arguments

transform	The transformation matrix to apply to the points [§2.3.9, page 28].
count	The number of points to transform.
points	A pointer to an array of 2D points to transform [§2.7.2, page 33].
points_return	A pointer to an array in which to store the transformed points [§2.7.2, page 33].

Returns

Zero if successful; otherwise, one of the following:

PEXBadHomoCoord	One or more of the transformed points has a homogeneous coordinate of 0.

Description

Invoke **PEXTransformPoints2D** to apply the specified homogeneous *transformation matrix* to the list of *points*.

When applying the transformation, the *points* are first converted to homogeneous points by assigning them a homogeneous coordinate of 1. The transformation is then applied:

$$P' = T \mathrm{x} P$$

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, then all points other than those with a homogeneous coordinate of zero are transformed and returned.

If the return array is the same as the input array, then this utility overwrites the input values with the transformed values.

Errors

None.

See Also

PEXTransformPoints, **PEXTransformPoints2DH, PEXTransformPoints4D**

PEXTransformPoints2DH

Synopsis

void **PEXTransformPoints2DH**(`PEXMatrix3x3` *transform*, int *count*, `PEXCoord` **points*, `PEXCoord` **points_return*)

Arguments

transform	The transformation matrix to apply to the points [§2.3.9, page 28].
count	The number of points to transform.
points	A pointer to an array of 2D homogeneous points to transform [§2.7.1, page 33].
points_return	A pointer to an array in which to store the transformed points [§2.7.1, page 33].

Returns

None.

Description

Invoke **PEXTransformPoints2DH** to apply the specified homogeneous transformation matrix to the list of two-dimensional homogeneous points ($P = x, y, w$). The transformation is applied:

$$P' = T\text{x}P$$

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, then this utility overwrites the input values with the transformed values.

Errors

None,

See Also

PEXTransformPoints, **PEXTransformPoints2D**, **PEXTransformPoints4D**

PEXTransformPoints4D

Synopsis

void **PEXTransformPoints4D**(`PEXMatrix` *transform*, int *count*, `PEXCoord4D` **points*, `PEXCoord4D` **points_return*)

Arguments

transform	The transformation matrix to apply to the points [§2.3.10, page 28].
count	The number of points to transform.
points	A pointer to an array of 4D points to transform [§2.7.3, page 33].
points_return	A pointer to an array in which to store the transformed points [§2.7.3, page 33].

Returns

None.

Description

Invoke **PEXTransformPoints4D** to apply the specified homogeneous *transformation matrix* to the list of 3D homogeneous *points* ($P = x, y, z, w$). The transformation is applied:

$$P' = T \text{x} P$$

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, then this utility overwrites the input values with the transformed values.

Errors

None.

See Also

PEXTransformPoints, **PEXTransformPoints2D**, **PEXTransformPoints2DH**

PEXTransformVectors

Synopsis

void **PEXTransformVectors**(`PEXMatrix` *transform*, int *count*, `PEXVector` **vectors*, `PEXVector` **vectors_return*)

Arguments

transform	The transformation matrix to apply to the vectors [§2.3.10, page 28].
count	The number of vectors to transform.
vectors	A pointer to the array of 3D vectors to transform [§2.31.1, page 67].
vectors_return	A pointer to an array in which to store the transformed vectors [§2.31.1, page 67].

Returns

None.

Description

Invoke **PEXTransformVectors** to apply the upper 3x3 submatrix of the specified *transformation matrix* to the list of three-dimensional *vectors*. The transformation is applied:

$$V' = T' \mathrm{x} V$$

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, then this utility overwrites the input values with the transformed values.

Errors

None.

See Also

PEXNormalizeVectors, **PEXTransformVectors2D**

PEXTransformVectors2D

Synopsis

void **PEXTransformVectors2D**(`PEXMatrix3x3` *transform*, int *count*, `PEXVector2D` **vectors*, `PEXVector2D` **vectors_return*)

Arguments

transform	The transformation matrix to apply to the vectors [§2.3.9, page 28].
count	The number of vectors to transform.
vectors	A pointer to the array of 2D vectors to transform [§2.31.2, page 67].
vectors_return	A pointer to an array in which to store the transformed vectors [§2.31.2, page 67].

Returns

None.

Description

Invoke **PEXTransformVectors2D** to apply the upper 2x2 submatrix of the specified *transformation matrix* to the list of two-dimensional *vectors*. The transformation is applied:

$$V' = T' \mathrm{x} V$$

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, then this function overwrites the input values with the transformed values.

Errors

None.

See Also

PEXNormalizeVectors2D, PEXTransformVectors

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 [§2.26.7, page 62].
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 [§2.26.3, page 62].
window_height	The height of the drawable.
transform_return	The returned transformation [§2.3.10, page 28].

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

Invoke **PEXXCToNPCTransform** compute 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 three-dimensional 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 [§2.26.7, page 62].
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 [§2.25.2, page 59].
window_height	The height of the drawable.
transform_return	The returned transformation [§2.3.9, page 28].

Returns

Zero if successful; otherwise, one of the following:

PEXBadViewport	xmin >= xmax, or ymin >= ymax
PEXBadSubVolume	xmin >= xmax, or ymin >= ymax, or zmin > zmax

Description

Invoke **PEXXCToNPCTransform2D** to compute the two-dimensional 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 two-dimensional components of the viewport-to-subvolume transformation, producing two-dimensional 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

25.4. Utilities for Computing Geometric Normals

These utilities compute the geometric normal vectors of the complex PEX output primitives. Their input arguments are identical to the arguments of the corresponding output command function. The facet-data return argument is filled in with the computed geometric normal of each facet.

FillArea
FillAreaSet
IndexedFillAreaSets
IndexedTriangles
PolyTriangle
QuadrilateralMesh
TriangleFan
Triangles
TriangleStrip

These utilities are augmented for PEXlib 5.2 to include the OCC form of the functions that were in PEXlib 5.1, allowing the three new PEXlib 5.2 data models to be used with these primitives. In addition, functions using the OCC form are provided for the new primitives. The non-OCC versions do not exist for these new primitives. Therefore, the functions appearing in PEXlib 5.1 have OCC counterparts specified on the same page.

A new "handedness" argument is added to the OCC forms of the functions for PEXlib 5.2. This argument allows you to generate normals for imported data that assumed a left-handed coordinate system, instead of needing to "flip" the normals yourself after they are generated. The non-OCC versions of the functions do not have the *handedness* argument and assume a right-handed coordinate system.

Use caution when using these utilities with fill areas that may possibly be concave, since it is possible to generate normals that point opposite to the intended direction if a concave portion of the fill area is used to generate the normal. In some cases, it may be possible to present the fill area vertices in a different order to obtain the intended result.

PEXOCCGeoNormFillArea

Synopsis

int **PEXOCCGeoNormFillArea**(`PEXOCC` *context*, `PEXPointer` *occ_facet_data*, unsigned int *count*, `PEXPointer` *occ_vertex_data*, int *handedness*)

int **PEXGeoNormFillArea**(unsigned int *facet_attributes*, unsigned int *vertex_attributes*, int *color_type*, `PEXFacetData` **facet_data*, unsigned int *count*, `PEXArrayOfVertex` *vertices*)

Arguments

context An OC context returned by a successful **PEXCreateOCC** call.

facet_attributes(OCC) A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.

vertex_attributes(OCC) A mask indicating the vertex attributes provided.

color_type (OCC) The type of color data provided.

facet_fp_data_count (OCC)
The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.

occ_facet_data A pointer to facet data. This function adds the geometric normal to this data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

facet_data A pointer to facet data. This function adds the geometric normal to this data [§2.12.12, page 37].

count The number of vertices.

occ_vertex_data A pointer to the vertex data defining the fill area [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

vertices An array of vertices defining the fill area [§2.32.87, page 79].

data_model (OCC) Specifies the data model used to represent the geometric data.

data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

handedness Orientation of the coordinate system assumed by the vertex data (**PEXVertexOrderRightHand**, **PEXVertexOrderLeftHand**).

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

Invoke **PEXOCCGeoNormFillArea** to compute the geometric normal of a fill area and store 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, then this utility returns unsuccessfully.

PEXGeoNormFillArea assumes the handedness is **PEXVertexOrderRightHand**.

Specify the *handedness* argument to generate normals that match the orientation of the coordinate system that the vertex data uses. **PEXVertexOrderRightHand** indicates that the coordinate system is right-handed, and **PEXVertexOrderLeftHand** indicates that the normals are to be "flipped" from the right-handed orientation to match a left-handed coordinate system.

If the facet attributes does not contain the bit **PEXGANormal**, then this utility does not compute the geometric normal. However, the utility still returns successfully.

Errors

None.

See Also

PEXOCCGeoNormFillAreaSet

Synopsis

int **PEXOCCGeoNormFillAreaSet**(`PEXOCC` *context*, unsigned int *count*, `PEXPointer` *occ_facet_data*, `PEXListOfVertexData` **occ_vertex_data*, int *handedness*)

int **PEXGeoNormFillAreaSet**(unsigned int *facet_attributes*, unsigned int *vertex_attributes*, int *color_type*, unsigned int *count*, `PEXFacetData` **facet_data*, `PEXListOfVertex` **vertex_lists*)

Arguments

context An OC context returned by a successful **PEXCreateOCC** call.

facet_attributes(OCC) A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.

vertex_attributes(OCC) A mask indicating the vertex attributes provided.

color_type (OCC) The type of color data provided.

facet_fp_data_count (OCC)
The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.

count The number of fill areas in the set.

occ_facet_data A pointer to facet data. This function adds the geometric normal to this data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

facet_data An array of facet data. This function adds the geometric normal to this data [§2.12.12, page 37].

occ_vertex_data A pointer to the list of vertex arrays defining each contour of the fill area set [§2.21.5, page 50]. See "Facet / Vertex Data Formats" on page 137.

vertex_lists A pointer to the list of vertex arrays defining each contour of the fill area set [§2.32.88, page 81].

data_model (OCC) Specifies the data model used to represent the geometric data.

data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

handedness Orientation of the coordinate system assumed by the vertex data (**PEXVertexOrderRightHand**, **PEXVertexOrderLeftHand**).

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

Invoke **PEXOCCGeoNormFillAreaSet** to compute the geometric normal of a fill area set and store 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.

PEXGeoNormFillAreaSet assumes the handedness is **PEXVertexOrderRightHand**.

Specify the *handedness* argument to generate normals that match the orientation of the coordinate system that the vertex data uses. **PEXVertexOrderRightHand** indicates that the coordinate system is right-handed, and **PEXVertexOrderLeftHand** indicates that the normals are to be "flipped" from the right-handed orientation to match a left-handed coordinate system.

If the facet attributes does not contain the bit **PEXGANormal**, then this utility does not compute the geometric normal. However, this utility still returns successfully.

Errors

None.

See Also

PEXOCCGeoNormIndexedFillAreaSets

Synopsis

int **PEXOCCGeoNormIndexedFillAreaSets**(`PEXOCC` *context*, unsigned int *set_count*, `PEXPointer` *occ_facet_data*, unsigned int *vertex_count*, `PEXPointer` *occ_vertex_data*, unsigned int *index_count*, `PEXConnectivityData` **connectivity*, int *handedness*)

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

context An OC context returned by a successful **PEXCreateOCC** call.

facet_attributes(OCC) A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.

vertex_attributes(OCC) A mask indicating the vertex attributes provided.

color_type (OCC) The type of color data provided.

facet_fp_data_count (OCC)
The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.

set_count The number of fill area sets.

occ_facet_data A pointer to facet data. This function adds the geometric normal to this data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

facet_data An array of facet data. This function adds the geometric normals to this data [§2.12.11, page 36].

vertex_count The number of vertices.

occ_vertex_data A pointer to the vertex data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

vertices An array of vertices [§2.32.87, page 79].

index_count The number of vertex connectivity indices.

connectivity A pointer to the list of contour connectivity data [§2.6.2, page 33].

data_model (OCC) Specifies the data model used to represent the geometric data.

data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

handedness Orientation of the coordinate system assumed by the vertex data (**PEXVertexOrderRightHand**, **PEXVertexOrderLeftHand**).

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

Invoke **PEXOCCGeoNormIndexedFillAreaSets** to compute the geometric normals of the set of fill area sets primitive and store 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 **PEXOCCGeoNormFillAreaSet**.

PEXGeoNormSetOfFillAreaSets assumes the handedness is **PEXVertexOrderRightHand**.

Specify the *handedness* argument to generate normals that match the orientation of the coordinate system that the vertex data uses. **PEXVertexOrderRightHand** indicates that the coordinate system is right-handed, and **PEXVertexOrderLeftHand** indicates that the normals are to be "flipped" from the right-handed orientation to match a left-handed coordinate system.

If the facet attributes does not contain the bit **PEXGANormal**, then the utility does not compute the geometric normal. However, the utility 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.

Errors

None.

See Also

PEXOCCGeoNormIndexedTriangles

Synopsis

int **PEXOCCGeoNormIndexedTriangles**(`PEXOCC` *context*, unsigned int *num_facets*, `PEXPointer` *occ_facet_data*, unsigned int *vertex_count*, `PEXPointer` *occ_vertex_data*, unsigned int *index_count*, `PEXConnectivityData` **connectivity*, int *handedness*)

Arguments

context An OC context returned by a successful **PEXCreateOCC** call.

facet_attributes(OCC) A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.

vertex_attributes(OCC) A mask indicating the vertex attributes provided.

color_type (OCC) The type of color data provided.

facet_fp_data_count (OCC)
The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.

num_facets The number of triangles.

occ_facet_data A pointer to facet data. This function adds the geometric normal to this data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

vertex_count The number of vertices.

occ_vertex_data A pointer to the vertex data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

index_count The number of vertex connectivity indices.

connectivity A pointer to the list of triangle connectivity data [§2.6.2, page 33].

data_model (OCC) Specifies the data model used to represent the geometric data.

data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

handedness Orientation of the coordinate system assumed by the vertex data (**PEXVertexOrderRightHand**, **PEXVertexOrderLeftHand**).

Returns

Zero if successful; otherwise, one of the following:

PEXBadPrimitive A normal cannot be computed for one or more triangles in the IndexedTriangles.

Description

Invoke **PEXOCCGeoNormIndexedTriangles** to compute the geometric normals of each of the triangles in this list of indexed independent triangles.and store 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:

$$Ng = (V1 \times V2) / |V1 \times V2|$$

A triangle is defined by three entries in the connectivity list, indexed by *i*, where the first vector (V1) is from the first point (P[connectivity_i]) of the triangle to the second point (P[$\text{connectivity}_{i+1}$]), and the second vector

(V2) is from the first point of the triangle to the third point (P[connectivity$_{i+2}$]).

Specify the *handedness* argument to generate normals that match the orientation of the coordinate system that the vertex data uses. **PEXVertexOrderRightHand** indicates that the coordinate system is right-handed, and **PEXVertexOrderLeftHand** indicates that the normals are to be "flipped" from the right-handed orientation to match a left-handed coordinate system.

If the facet attributes does not contain the bit **PEXGANormal**, then this utility does not compute the geometric normal. However, this utility 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 primitive.

Errors

None.

See Also

PEXOCCGeoNormPolyTriangle

Synopsis

int **PEXOCCGeoNormPolyTriangle**(`PEXOCC` *context*, `PEXPointer` *occ_facet_data*, unsigned int *count*, `PEXPointer` *occ_vertex_data*, int *handedness*)

Arguments

context An OC context returned by a successful **PEXCreateOCC** call.

facet_attributes(OCC) A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.

vertex_attributes(OCC) A mask indicating the vertex attributes provided.

color_type (OCC) The type of color data provided.

facet_fp_data_count (OCC)
The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.

occ_facet_data A pointer to facet data. This function adds the geometric normal to this data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

count The number of vertices.

occ_vertex_data A pointer to the vertex data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

data_model (OCC) Specifies the data model used to represent the geometric data.

data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

handedness Orientation of the coordinate system assumed by the vertex data (**PEXVertexOrderRightHand**, **PEXVertexOrderLeftHand**).

Returns

Zero if successful; otherwise, one of the following:

PEXBadPrimitive A normal cannot be computed for one or more triangles in the PolyTriangle.

Description

Invoke **PEXOCCGeoNormPolyTriangle** to compute the geometric normals of each of the triangles in a PolyTriangle and store 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:

$$Ng = (V1 \times V2) / |V1 \times V2|$$

See the description in “PEXOCCPolyTriangle - Generalized Triangle Primitive (5.2)” on page 223 to determine how the vectors V1 and V2 are determined.

Specify the *handedness* argument to generate normals that match the orientation of the coordinate system that the vertex data uses. **PEXVertexOrderRightHand** indicates that the coordinate system is right-handed, and **PEXVertexOrderLeftHand** indicates that the normals are to be “flipped” from the right-handed orientation to match a left-handed coordinate system.

If the facet attributes does not contain the bit **PEXGANormal**, then this utility does not compute the geometric normal. However, this utility 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 list.

Errors

None.

See Also

PEXOCCGeoNormQuadrilateralMesh

Synopsis

int **PEXOCCGeoNormQuadrilateralMesh**(`PEXOCC` *context*, `PEXPointer` *occ_facet_data*, unsigned int *col_count*, unsigned int *row_count*, `PEXPointer` *occ_vertices*, int *handedness*)

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

context An OC context returned by a successful **PEXCreateOCC** call.

facet_attributes(OCC) A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.

vertex_attributes(OCC) A mask indicating the vertex attributes provided.

color_type (OCC) The type of color data provided.

facet_fp_data_count (OCC)
The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.

occ_facet_data A pointer to facet data. This function adds the geometric normal to this data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

facet_data An array of facet data. This function adds the geometric normals to this data [§2.12.11, page 36].

col_count The number of columns in the vertex array.

row_count The number of rows in the vertex array.

occ_vertex_data A pointer a the vertex data which is interpreted as a two-dimensional (row-major) array defining the quadrilateral mesh [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

vertices A two-dimensional (row-major) array of vertices defining the quadrilateral mesh [§2.32.87, page 79].

data_model (OCC) Specifies the data model used to represent the geometric data.

data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

handedness Orientation of the coordinate system assumed by the vertex data (**PEXVertexOrderRightHand**, **PEXVertexOrderLeftHand**).

Returns

Zero if successful; otherwise, one of the following:

PEXBadPrimitive A normal cannot be computed for one or more quadrilaterals in the mesh.

Description

Invoke **PEXOCCGeoNormQuadrilateralMesh** to compute the geometric normals of a quadrilateral mesh and store 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:

$$Ng = (V1 \times V2) / |V1 \times V2|$$

Given the quadrilateral composed of four vertices, $P_{i,j}$, where *i* indicates the row of the point and *j* its column, the first vector, V1, is from $P_{i,j}$, to $P_{i+1,j+1}$. The second vector, V2, is from $P_{i+1,j}$ to $P_{i,j+1}$.

PEXGeoNormQuadrilateralMesh assumes the handedness is **PEXVertexOrderRightHand**.

Specify the *handedness* argument to generate normals that match the orientation of the coordinate system that the vertex data uses. **PEXVertexOrderRightHand** indicates that the coordinate system is right-handed, and **PEXVertexOrderLeftHand** indicates that the normals are to be "flipped" from the right-handed orientation to match a left-handed coordinate system.

If the facet attributes does not contain the bit **PEXGANormal**, then this utility does not compute the geometric normal. However, this utility 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

PEXOCCGeoNormTriangleFan

Synopsis

int **PEXOCCGeoNormTriangleFan**(`PEXOCC` *context*, `PEXPointer` *occ_facet_data*, unsigned int *count*, `PEXPointer` *occ_vertex_data*, int *handedness*)

Arguments

context — An OC context returned by a successful **PEXCreateOCC** call.

facet_attributes(OCC) — A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.

vertex_attributes(OCC) — A mask indicating the vertex attributes provided.

color_type (OCC) — The type of color data provided.

facet_fp_data_count (OCC) — The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC) — The number of optional floating-point numbers for each vertex.

occ_facet_data — A pointer to facet data. This function adds the geometric normal to this data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

count — The number of vertices.

occ_vertex_data — A pointer to the vertex data [§2.18.1, page 46]. See “Facet / Vertex Data Formats” on page 137.

data_model (OCC) — Specifies the data model used to represent the geometric data.

data_model_specs (OCC) — Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

handedness — Orientation of the coordinate system assumed by the vertex data (**PEXVertexOrderRightHand**, **PEXVertexOrderLeftHand**).

Returns

Zero if successful; otherwise, one of the following:

PEXBadPrimitive — A normal cannot be computed for one or more triangles in the fan.

Description

Invoke **PEXOCCGeoNormTriangleFan** to compute the geometric normals of a triangle fan and store 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:

$$Ng = (V1 \times V2) / |V1 \times V2|$$

For each triangle, *i*, the first vector (V1) is from the first point (P_1) of the triangle fan (the common vertex for the entire fan) to the second point (P_{i+1}), and the second vector (V2) is from the first point (P_1) of the triangle fan to the third point (P_{i+2}).

Specify the *handedness* argument to generate normals that match the orientation of the coordinate system that the vertex data uses. **PEXVertexOrderRightHand** indicates that the coordinate system is right-handed, and **PEXVertexOrderLeftHand** indicates that the normals are to be “flipped” from the right-handed orientation to match a left-handed coordinate system.

If the facet attributes does not contain the bit **PEXGANormal**, then this utility does not compute the geometric normal. However, this utility 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 fan.

Errors

None.

See Also

PEXOCCGeoNormTriangles

Synopsis

int **PEXOCCGeoNormTriangles**(`PEXOCC` *context*, `PEXPointer` *occ_facet_data*, unsigned int *count*, `PEXPointer` *occ_vertex_data*, int *handedness*)

Arguments

context An OC context returned by a successful **PEXCreateOCC** call.

facet_attributes(OCC) A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.

vertex_attributes(OCC) A mask indicating the vertex attributes provided.

color_type (OCC) The type of color data provided.

facet_fp_data_count (OCC)
The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.

occ_facet_data A pointer to facet data. This function adds the geometric normal to this data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

count The number of vertices.

occ_vertex_data A pointer to the vertex data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

data_model (OCC) Specifies the data model used to represent the geometric data.

data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

handedness Orientation of the coordinate system assumed by the vertex data (**PEXVertexOrderRightHand**, **PEXVertexOrderLeftHand**).

Returns

Zero if successful; otherwise, one of the following:

PEXBadPrimitive A normal cannot be computed for one or more triangles in the list.

Description

Invoke **PEXOCCGeoNormTriangles** to compute the geometric normals of each of the triangles in an independent triangle list and store 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:

$$Ng = (V1 \times V2) / |V1 \times V2|$$

For each triangle, *i* (*i*=0 for the first triangle), the first vector (V1) is from the first point (P_{3i}) of the triangle to the second point (P_{3i+1}), and the second vector (V2) is from the first point (P_{3i}) of the triangle to the third point (P_{3i+2}).

Specify the *handedness* argument to generate normals that match the orientation of the coordinate system that the vertex data uses. **PEXVertexOrderRightHand** indicates that the coordinate system is right-handed, and **PEXVertexOrderLeftHand** indicates that the normals are to be "flipped" from the right-handed orientation to match a left-handed coordinate system.

If the facet attributes does not contain the bit **PEXGANormal**, then this utility does not compute the geometric normal. However, this utility 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 list.

Errors

None.

See Also

PEXOCCGeoNormTriangleStrip

Synopsis

int **PEXOCCGeoNormTriangleStrip**(`PEXOCC` *context*, `PEXPointer` *occ_facet_data*, unsigned int *count*, `PEXPointer` *occ_vertex_data*, int *handedness*)

int **PEXGeoNormTriangleStrip**(unsigned int *facet_attributes*, unsigned int *vertex_attributes*, int *color_type*, `PEXArrayOfFacetData` *facet_data*, unsigned int *count*, `PEXArrayOfVertex` *vertices*)

Arguments

context An OC context returned by a successful **PEXCreateOCC** call.

facet_attributes(OCC) A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.

vertex_attributes(OCC) A mask indicating the vertex attributes provided.

color_type (OCC) The type of color data provided.

facet_fp_data_count (OCC) The number of optional floating-point numbers for each facet.

surface_vertex_fp_data_count (OCC) The number of optional floating-point numbers for each vertex.

occ_facet_data A pointer to facet data. This function adds the geometric normal to this data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

facet_data An array of facet data. This function adds the geometric normals to this data [§2.12.11, page 36].

count The number of vertices.

occ_vertex_data A pointer to the vertex data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

vertices An array of vertices defining the triangle strip [§2.32.87, page 79].

data_model (OCC) Specifies the data model used to represent the geometric data.

data_model_specs (OCC) Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

handedness Orientation of the coordinate system assumed by the vertex data (**PEXVertexOrderRightHand**, **PEXVertexOrderLeftHand**).

Returns

Zero if successful; otherwise, one of the following:

PEXBadPrimitive A normal cannot be computed for one or more triangles in the strip.

Description

Invoke **PEXOCCGeoNormTriangleStrip** to compute the geometric normals of a triangle strip and store 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:

$$Ng = (V1 \times V2) / |V1 \times V2|$$

For the first, third, and subsequent odd-numbered triangles, the first vector (V1) is from the first point (P_i) of the triangle to the second point (P_{i+1}), and the second vector (V2) is from the first point of the triangle to the third point (P_{i+2}). For the second, fourth, and subsequent even-numbered triangles, the first vector is from the first point (P_i) of the triangle to the third point (P_{i+2}), and the second vector is from the first point of the triangle to the second point (P_{i+1}).

PEXGeoNormTriangleStrip assumes the handedness is **PEXVertexOrderRightHand**.

Specify the *handedness* argument to generate normals that match the orientation of the coordinate system that the vertex data uses. **PEXVertexOrderRightHand** indicates that the coordinate system is right-handed, and **PEXVertexOrderLeftHand** indicates that the normals are to be "flipped" from the right-handed orientation to match a left-handed coordinate system.

If the facet attributes does not contain the bit **PEXGANormal**, then this utility does not compute the geometric normal. However, this utility 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

25.5. Utilities for Computing Shape Hints

These utilities compute a shape hint for the fill area and quadrilateral mesh PEX output primitives. These utilities are intended for you to use to determine a shape hint for your data, if not already known, and then pass the hint to the implementation. Many implementations use the shape hint to optimize the rendering of these primitives, making it worthwhile to provide an accurate shape hint.

FillArea
FillAreaSet
IndexedFillAreaSets
QuadrilateralMesh

PEXOCCComputeFillAreaShapeHint

Synopsis

int **PEXOCCComputeFillAreaShapeHint**(`PEXOCC` *context*, unsigned int *count*, `PEXPointer` *vertex_data*)

Arguments

context	An OC context returned by a successful **PEXCreateOCC** call.
vertex_attributes(OCC)	A mask indicating the vertex attributes provided.
color_type (OCC)	The type of color data provided.
surface_vertex_fp_data_count (OCC)	The number of optional floating-point numbers for each vertex.
count	The number of vertices.
*vert*ex_data	A pointer to the vertex data defining the fill area [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.
data_model (OCC)	Specifies the data model used to represent the geometric data.
data_model_specs (OCC)	Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

Shape hint; one of the following: **PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex**, **PEXShapeUnknown**.

Description

Invoke **PEXOCCComputeFillAreaShapeHint** to compute the shape hint of a fill area and return it as the function return value.

If this function cannot determine a shape hint because of invalid or degenerate vertex data, the function will return the least optimistic hint, **PEXShapeUnknown**. This function does not perform a planarity check, so it returns a hint that may be invalid for non-planar fill areas.

Errors

None.

See Also

PEXOCCComputeFillAreaSetShapeHint

Synopsis

int **PEXOCCComputeFillAreaSetShapeHint**(`PEXOCC` *context*, unsigned int *count*,
`PEXListOfVertexData` **vertex_lists*)

Arguments

context An OC context returned by a successful **PEXCreateOCC** call.
vertex_attributes(OCC) A mask indicating the vertex attributes provided.
color_type (OCC) The type of color data provided.
surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.
count The number of fill areas in the set.
vertex_lists A pointer to the list of vertex arrays defining each contour of the fill area set [§2.21.5, page 50]. See “Facet / Vertex Data Formats” on page 137.
data_model (OCC) Specifies the data model used to represent the geometric data.
data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

Shape hint; one of the following: **PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex**, **PEXShapeUnknown**.

Description

Invoke **PEXOCCComputeFillAreaSetShapeHint** to compute the shape hint for all fill areas of a fill area set and return it as the function return value.

If this function cannot determine a shape hint because of invalid or degenerate vertex data, the function will return the least optimistic hint, **PEXShapeUnknown**. If the shape hint for any two fill areas in the set is different, the function returns **PEXShapeUnknown**. This function does not perform a planarity check, so it returns a hint that may be invalid for non-planar fill area sets. This function does not compute a contour hint.

Errors

None.

See Also

PEXOCCComputeIndexedFillAreaSetsShapeHint

Synopsis

int **PEXOCCComputeIndexedFillAreaSetsShapeHint**(`PEXOCC` *context*, unsigned int *set_count*, unsigned int *vertex_count*, `PEXPointer` *vertex_data*, unsigned int *index_count*, `PEXConnectivityData` **connectivity*)

Arguments

context An OC context returned by a successful **PEXCreateOCC** call.
vertex_attributes(OCC) A mask indicating the vertex attributes provided.
color_type (OCC) The type of color data provided.
surface_vertex_fp_data_count (OCC)
The number of optional floating-point numbers for each vertex.
set_count The number of fill area sets.
vertex_count The number of vertices.
vertex_data A pointer to the vertex data [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.
index_count The number of vertex connectivity indices.
connectivity A pointer to the list of contour connectivity data [§2.6.2, page 33].
data_model (OCC) Specifies the data model used to represent the geometric data.
data_model_specs (OCC)
Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

Shape hint; one of the following: **PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex**, **PEXShapeUnknown**.

Description

Invoke **PEXOCCComputeIndexedFillAreaSetsShapeHint** to compute the shape hint for all fill areas of an indexed fill area sets primitive and return it as the function return value.

If this function cannot determine a shape hint because of invalid or degenerate vertex data, the function will return the least optimistic hint, **PEXShapeUnknown**. If the shape hint for any two fill areas in all the fill area sets is different, the function returns **PEXShapeUnknown**. This function does not perform a planarity check, so it returns a hint that may be invalid for non-planar fill area sets. This function does not compute a contour hint.

Errors

None.

See Also

PEXOCCComputeQuadrilateralMeshShapeHint

Synopsis

int **PEXOCCComputeQuadrilateralMeshShapeHint**(`PEXOCC` *context*, unsigned int *col_count*, unsigned int *row_count*, `PEXPointer` *vertex_data*)

Arguments

context — An OC context returned by a successful **PEXCreateOCC** call.

vertex_attributes(OCC) — A mask indicating the vertex attributes provided.

color_type (OCC) — The type of color data provided.

surface_vertex_fp_data_count (OCC) — The number of optional floating-point numbers for each vertex.

col_count — The number of columns in the vertex array.

row_count — The number of rows in the vertex array.

vertex_data — A pointer a the vertex data which is interpreted as a two-dimensional (row-major) array defining the quadrilateral mesh [§2.18.1, page 46]. See "Facet / Vertex Data Formats" on page 137.

data_model (OCC) — Specifies the data model used to represent the geometric data.

data_model_specs (OCC) — Offset and size data for the **PEXDataStride** or **PEXDataUnpacked** data models.

Returns

Shape hint; one of the following: **PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex**, **PEXShapeUnknown**.

Description

Invoke **PEXOCCComputeQuadrilateralMeshShapeHint** to compute the shape hint for all quads of a quadrilateral mesh and return it as the function return value.

If this function cannot determine a shape hint because of invalid or degenerate vertex data, the function will return the least optimistic hint, **PEXShapeUnknown**. If the shape hint for any two quads in the mesh is different, the function returns **PEXShapeUnknown**. This function does not perform a planarity check, so it returns a hint that may be invalid for non-planar fill area sets.

Errors

None.

See Also

Appendix A

PEXlib Programming and PEX Protocol Levels

A.1. Introduction

You must make several important decisions when creating a PEXlib application that may encounter a variety of PEX protocol levels. If your application operates in a distributed heterogeneous environment, it is likely that the PEX servers in the network may support different protocol levels due simply to the timing of vendor releases.

This discussion only address different levels of the PEX Version 5 protocol. PEXlib 5.x will not operate with PEX version 4, for example.

You must decide what version of PEXlib to use (5.1 or 5.2) and what levels of the PEX protocol your application will work with (5.0, 5.1, or 5.2). The most straightforward path is to simply choose PEXlib 5.2 and restrict the operation of the application to servers that support only PEX 5.2. However, if your application does not rely heavily on PEX 5.2 functionality, it may be possible to code multiple paths to allow it to work with multiple protocol levels.

A.1.1. Libraries

First, a quick word about libraries. If your application is written to PEXlib 5.1 specifications, then it also follows PEXlib 5.2 specifications. You are able to compile and link the same code with PEXlib 5.2 header files and libraries.

If you wish to use features that are unique to PEX 5.2, you must compile with PEXlib 5.2 libraries in order to access these PEX 5.2 functions conveniently.

It is also possible for you to use PEXlib 5.2 to access only PEX 5.1 or PEX 5.0 functionality. You may want to do this to take advantage of PEXlib 5.2 features like the OC Context for accessing older PEX function while remaining interoperable with older servers. Much of the rest of this appendix is devoted to using PEXlib in this fashion.

A.2. 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. The information is available in a data structure returned by the **PEXInitialize** function, or by using the **PEXGetExtensionInfo** function:

```
int                 status;
PEXExtensionInfo    *pexinfo;
char                errstr[PEXErrorStringLength];

status = PEXInitialize( display, &pexinfo, PEXErrorStringLength, errstr );
if (status != 0) {
     fprintf( stderr, "%s\n", errstr );
     exit(0);
}
/* pexinfo->major - check to be sure it is 5 */
/* pexinfo->minor - check for 0, 1, 2, ....  */

/* You can also obtain this info at any time by calling: */

pexinfo = PEXGetExtensionInfo( display );
```

Now that you know what version of the PEX protocol the server supports, here is an example of how you write code to deal with different levels:

```
if (pexinfo->minor >= 2)
      PEXOCCSetInteriorStyle(context, PEXInteriorStyleTexture);
else
      PEXOCCSetInteriorStyle(context, PEXInteriorStyleSolid);
```

This example simply set the interior style attribute of the pipeline state to the texturing state if the PEX server implementation is 5.2 or higher. The code uses “>=” because future versions of PEX will continue to support features that are currently defined. If the server is a 5.0 or 5.1 server, then this code uses the “fall-back” of telling the server to use the solid interior style.

Note that there is an important distinction between a feature being *defined*, as opposed to being *supported*. PEX 5.0 and PEX 5.1 do not define PEXInteriorStyleTexture, so if you send this value to a 5.0 or 5.1 server, you get a protocol error. In contrast, if you send this value to a 5.2 server that does not support this interior style, the implementation quietly uses the default interior style.

When the application can afford to use fall-backs as in the above example, you are able to code multiple paths that allow your application to work, to some extent, on older servers.

A.3. Summary of changes for PEX 5.1 and PEX 5.2.

Below is a list containing features and at what PEX protocol level they are introduced. You should not try to use a feature with a PEX server with a version less than the version at which the feature was defined.

All other features are defined in PEX 5.0 and may be used in all later versions of PEX.

A.3.1. Enumerated Types

Applications should not attempt to query undefined enum types.

PEXETPickOneMethod - (5.1)

PEXETPickAllMethod - (5.1)

PEXETEscape - (5.1)

PEXETAlphaBlendFunction -(5.2)

PEXETDataMapMethod -(5.2)

PEXETEchoMethod -(5.2)

PEXETHighlightMethod -(5.2)

PEXETHLHSRIdentifier -(5.2)

PEXETLineCapStyle -(5.2)

PEXETLineJoinStyle -(5.2)

PEXETMarkerGlyphType -(5.2)

PEXETPrimitiveAABlendFunction -(5.2)

PEXETSupportedTexelResources -(5.2)

PEXETTMBoundaryClampMethod -(5.2)

PEXETTMBoundaryCondition - (5.2)

PEXETTMCompositionMethod -(5.2)

PEXETTMDimension - (5.2)

PEXETTMParameterizationMethod - (5.2)

PEXETTMPerspectiveCorrection -(5.2)

PEXETTMRenderingControl -(5.2)

PEXETTMResourceHints -(5.2)

PEXETTMSampleFrequency -(5.2)

PEXETTMSampleMethod -(5.2)

PEXETTMTexelType -(5.2)

PEXETTransparencyMethod -(5.2)

PEXETZBufferTest -(5.2)

A.3.2. Enumerated Type Values

Applications should not expect to find the undefined values in the associated enum types

PEXETHLHSRMode - **PEXHLHSRZBufferID (5.1)**

PEXETColorType - PEXColorTypeRGBA, PEXColorTypeRGBA8, PEXColorTypeRGBA16(5.2)

PEXETAlphaBlendFunction - All Values **(5.2)**

PEXETDataMapMethod - All Values**(5.2)**

PEXETEchoMethod - All Values**(5.2)**

PEXETHatchStyle - PEXHatchStyle45Degrees, PEXHatchStyle135Degrees(5.2)

PEXETHighlightMethod - All Values**(5.2)**

PEXETHLHSRIdentifier - All Values**(5.2)**

PEXETInteriorStyle - PEXInteriorStyleTexture(5.2)

PEXETLineCapStyle - All Values**(5.2)**

PEXETLineJoinStyle - All Values**(5.2)**

PEXETLineType - PEXLineTypeCenterLine, PEXLineTypePhantom(5.2)

PEXETMarkerGlyphType - All Values**(5.2)**

PEXETPrimitiveAABlendFunction - All Values**(5.2)**

PEXETSupportedTexelResources - All Values**(5.2)**

PEXETTMBoundaryClampMethod - All Values**(5.2)**

PEXETTMBoundaryCondition - All Values**(5.2)**

PEXETTMCompositionMethod - All Values**(5.2)**

PEXETTMDimension - All Values **(5.2)**

PEXETTMParameterizationMethod - All Values**(5.2)**

PEXETTMPerspectiveCorrection - All Values**(5.2)**

PEXETTMRenderingControl - All Values**(5.2)**

PEXETTMResourceHints - All Values**(5.2)**

PEXETTMSampleFrequency - All Values**(5.2)**

PEXETTMSampleMethod - All Values**(5.2)**

PEXETTMTexelType - All Values**(5.2)**

PEXETTransparencyMethod - All Values**(5.2)**

PEXETZBufferTest - All Values**(5.2)**

A.3.3. PEX Errors

Applications so not need to handle undefined errors.

BadPEXStructurePermission(5.2)

A.3.4. PEX Events

Applications do not need to watch for undefined events.

PEXMaxHitsReached - **(5.1)**

A.3.5. Implementation-Dependent Constants

Applications should not attempt to query undefined implementation-dependent constants.

PEXIDMaxHitsEventSupported - **(5.1)**

PEXIDConcentrationClampSupported -(5.2)

PEXIDDitheringControllable - (5.2)

PEXIDEyepointUsed -(5.2)

PEXIDMaxMarkerVertices -(5.2)

PEXIDMaxNumPickResults -(5.2)

PEXIDPrimitiveAASupported -(5.2)

PEXIDQuadraticAttenSupported -(5.2)

PEXIDPixelColorSupported -(5.2)

PEXIDMaxTextureMaps -(5.2)

PEXIDMaxFastTMSize -(5.2)

PEXIDPowerOfTwoTMSizesRequired -(5.2)

PEXIDSquareTMRequired -(5.2)

PEXIDSamplingHintsUsed -(5.2)

A.3.6. General Functions

Applications should not use undefined functions (requests).

PEXMatchRenderingTargets - (5.1)

PEXEscape -(5.1)

PEXEscapeWithReply - (5.1)

PEXSetEchoColor - **(5.1)**

PEXQueryColorApproximation -(5.2)

A.3.7. Output Commands

Applications should not render undefined output commands and should not store undefined output commands in a structure. Some output commands have had some incremental new function added to them for a particular level of PEX. This additional functionality is indicated below inside parenthesis.

PEXNoop - **(5.1)**

PEXOCCArcAreas -(5.2)

PEXOCCArcs -(5.2)

PEXOCCCones -(5.2)

PEXOCCIndexedTriangles- (5.2)

PEXOCCMarkers (with data) -(5.2)

PEXOCCNURBCurve (with data) -(5.2)

PEXOCCNURBSurface (with data) -(5.2)

PEXOCCPolylines -(5.2)

PEXOCCPolyTriangles -(5.2)

PEXOCCSpheres -(5.2)

PEXOCCTriangleFan -(5.2)

PEXOCCTriangles -(5.2)

PEXOCCSetAlphaBlendFunction -(5.2)

PEXOCCSetBFDataMappingIndex -(5.2)

PEXOCCSetBFDataMappingMethod -(5.2)

PEXOCCSetBFInteriorBundleIndex -(5.2)

PEXOCCSetBFReflectanceBundleIndex -(5.2)

PEXOCCSetBFSurfaceActiveTextures -(5.2)

PEXOCCSetDataMappingIndex -(5.2)

PEXOCCSetDataMappingMethod -(5.2)

PEXOCCSetDrawingFunction -(5.2)

PEXOCCSetHighlightIndex -(5.2)

PEXOCCSetGlobalTransform (matrix hints) -(5.2)

PEXOCCSetLineCapStyle -(5.2)

PEXOCCSetLineJoinStyle -(5.2)

PEXOCCSetLineMiterLimit -(5.2)

PEXOCCSetLocalTransform (matrix hints) -(5.2)

PEXOCCSetParametricSurfaceBundleIndex -(5.2)

PEXOCCSetPlaneMask -(5.2)

PEXOCCSetPrimitiveAntialiasing -(5.2)

PEXOCCSetReflectanceBundleIndex -(5.2)

PEXOCCSetSurfaceActiveTextures -(5.2)

PEXOCCSetTMPerspectiveCorrection -(5.2)

PEXOCCSetTMResourceHints -(5.2)

PEXOCCSetTMSampleFrequency -(5.2)

PEXOCCSetVertexOrder -(5.2)

PEXOCCSetZBufferTest -(5.2)

PEXOCCConditionFlags -(5.2)

PEXOCCConditionalExecuteStructure -(5.2)

PEXOCCConditionalIncludeStructure -(5.2)

PEXOCCConditionalReturn -(5.2)

PEXOCCIncludeStructure -(5.2)

PEXOCCSequentialProcessing -(5.2)

PEXOCCTestExtent -(5.2)

PEXOCCTextExtent2D -(5.2)

A.3.8. Lookup Tables

Applications should not use undefined lookup tables. Some lookup tables have new additional fields defined for a given release of PEX, indicated by parenthesis.

PEXLUTDataMapping -(5.2)

PEXLUTExtentSize -(5.2)

PEXLUTHighlight -(5.2)

PEXLUTInteriorBundle (parametric surface characteristics) -(5.2)

PEXLUTLight (concentration clamp region, quadratic attenuation factor) -(5.2)

PEXLUTLineBundle (line cap style, line join style) -(5.2)

PEXLUTMarkerGlyph -(5.2)

PEXLUTTextureMapping -(5.2)

PEXLUTView (eyepoint) -(5.2)

A.3.9. Lookup Table Functions

Applications should not use undefined functions (requests).

PEXChangeTableValues -(5.2)

PEXGetTableValues -(5.2)

A.3.10. Pipeline Context

Applications should not access undefined fields in the pipeline context

PEXPCActiveTextures -(5.2)

PEXPCAlphaBlendFunction -(5.2)

PEXPCBFActiveTextures -(5.2)

PEXPCBFDataMappingIndex -(5.2)

PEXPCBFDataMappingMethod -(5.2)

PEXPCBFInteriorBundleIndex -(5.2)

PEXPCBFReflectanceBundleIndex -(5.2)

PEXPCConditionFlags -(5.2)

PEXPCDataMappingIndex -(5.2)

PEXPCDataMappingMethod -(5.2)

PEXPCDrawingFunction -(5.2)

PEXPCGlobalTransformHints -(5.2)

PEXPCHighlightIndex -(5.2)

PEXPCLineCapStyle -(5.2)

PEXPCLineJoinStyle -(5.2)

PEXPCLineMiterLimit -(5.2)

PEXPCLocalTransformHints -(5.2)

PEXPCMoreASFValues -(5.2)

PEXPCParaSurfBundleIndex -(5.2)

PEXPCPlaneMask -(5.2)

PEXPCPrimAntialiasing -(5.2)

PEXPCSequentialProcessing -(5.2)

PEXPCTMPerspectiveCorrection -(5.2)

PEXPCTMResourceHints -(5.2)

PEXPCTMSampleFrequency -(5.2)

PEXPCVertexOrder -(5.2)

PEXPCZBufferTest -(5.2)

A.3.11. Renderer Functions

Applications should not use undefined functions (requests).

PEXRenderElements -(5.1)

PEXAccumulateState - (5.1)

PEXAccumulateState -(5.2)

PEXClearRenderer -(5.2)

PEXCopyAlphaBufferToDrawable -(5.2)

PEXCopyDrawableToAlphaBuffer -(5.2)

PEXCopyDrawableToZBuffer -(5.2)

PEXCopyPCToPipelineState -(5.2)

PEXCopyPipelineStateToPC -(5.2)

PEXCopyZBufferToDrawable -(5.2)

PEXFlushRenderer -(5.2)

PEXGetZBuffer -(5.2)

PEXInitMultipass -(5.2)

PEXInitRenderer -(5.2)

PEXNextPass -(5.2)

PEXNextPassWithoutReply -(5.2)

PEXPutZBuffer -(5.2)

A.3.12. Renderer Dynamics

Applications should not query undefined renderer dynamics bits. Note: For PEX 5.2 and later, all renderer attributes are considered dynamic, so PEX 5.2 applications should have no need to query this information.

PEXRDNPickNameSet - (5.1)

PEXRDNPickNameSetContents - (5.1)

PEXRDAEchoMode - (5.1)

A.3.13. Renderer Attributes

Applications should not set or query undefined renderer attributes.

PEXRXAPickIncl -(5.1)

PEXRXAPickExcl - (5.1)

PEXRXAPickStartPath -(5.1)

PEXRXABackgroundColor -(5.1)

PEXRXAClearImage -(5.1)

PEXRXAClearZ -(5.1)

PEXRXAEchoMode -(5.1)

PEXRXAClearZ -(5.2)

PEXRXAClearAlpha -(5.2)

PEXRXAPHIGSInteriorBundle -(5.2)

PEXRXAExtentSizeTable -(5.2)

PEXRXADataMappingTable -(5.2)

PEXRXAHighlightTable -(5.2)

PEXRXAMarkerGlyphTable -(5.2)

PEXRXATransparencyMethod -(5.2)

PEXRXATransparencyThresholds -(5.2)

PEXRXAPixelColorMode -(5.2)

PEXRXAEchoColor -(5.2)

PEXRXAEchoMethod -(5.2)

PEXRXATextureMappingTable -(5.2)

A.3.14. Renderer Picking Functions

Applications should not use undefined functions (requests).

PEXPickOne -(5.1)

PEXBeginPickOne -(5.1)

PEXEndPickOne -(5.1)

PEXPickAll -(5.1)

PEXBeginPickAll -(5.1)

PEXEndPickAll -(5.1)

A.3.15. Structure Operations

Applications should not use undefined functions (requests).

PEXSetElementPtrAtPickID -(5.2)

PEXSetStructurePermission -(5.2)

A.3.16. Texture Mapping Resource Functions

Applications should not use undefined functions (requests).

PEXCreateMipmapTM -(5.2)

PEXCreateMipmapTMFromResources -(5.2)

A.3.17. Accumulation Buffer Functions

Applications should not use undefined functions (requests).

PEXAccumulateBuffer -(5.2)

PEXAllocateAccumulationBuffer -(5.2)

PEXFreeAccumulationBuffer -(5.2)

PEXLoadAccumulationBuffer -(5.2)

PEXReturnAccumulationBuffer -(5.2)

Appendix B

PEX Functional Overview

(This is an excerpt from the PEX 5.2 specification, replicated here for convenience. Page references have been modified to refer to most closely related information found in PEXlib; however, for the actual PEX information, you should obtain a copy of the PEX specification).

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

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

PEX Components

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

2.1. Output Commands

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

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

2.1.1. Primitives

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

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

PEX has a set of defined primitive types. These primitives can be classified into the four broad areas described below.

2.1.1.1. Surface Primitives

Surface primitives are formed of one or more facets. A facet is formed by a set of regions in a single plane. Each region

1. International Standard ISO/IEC 9592-1, Programmer's Hierarchical Interactive Graphics System (PHIGS).
2. International Standard ISO/IEC 9492-4, Programmer's Hierarchical Interactive Graphics System Plus Lumière und Surfaces (PHIGS PLUS).

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

The surface primitives are listed in the section "Surfaces - General Attributes" on page 164.

- Interior and Edges

  The interior of a facet is defined by the even-odd rule (§PHIGS 4.5.1) and includes the region underneath the edges of the facet. There are separate surface interior and surface edge attributes. The rendering of edges is enabled by the pipeline state's *surface edges* field, modified with the Surface Edge Flag OC (page 395). Some surface primitives optionally include a visibility flag for each edge of the primitive. When edge rendering is enabled, these individually specified edge visibility flags are used if they are available. Otherwise all of a primitive's edges are rendered when edge rendering is enabled.

- Geometric Normal

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

  How a facet's geometric normal is determined is a function of the type of surface primitive. For the PEX surface primitives which are identical to PHIGS primitives, this determination is defined in PHIGS PLUS 4.5.5.2. Notice if the geometric normal is computed, the order in which the facet's vertices are defined plays a role. The pipeline state's *vertex order* field, modified with the Vertex Order OC (page 412), allows the ordering, for the purpose of geometric normal computation, to be reversed.

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

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

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

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

- Facet Distinguishing and Culling (§PHIGS PLUS 4.5.6)

  In addition to the surface interior attributes, there are back-facing surface interior attributes. The back-facing attributes are applied to back-facing facets only if enabled by the pipeline state's *facet distinguish flag* field, modified with the Facet Distinguish Flag OC (page 315); otherwise the surface interior attributes are applied to both front- and back-facing facets. Therefore facet distinguishing indicates whether back-facing facets are distinguishable by being rendered with different surface interior attributes than the front-facing facets.

  Facet culling allows facets to be thrown out, that is not rendered, based on whether the facets are front-facing or back-facing. Whether front-facing, back-facing, or no facets are culled is controlled by the pipeline state's *facet culling mode* field, modified with the Facet Culling Mode OC (page 314). Facet culling is independent of facet distinguishing.

2.1.1.2. Line Primitives

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

The line primitives are listed in the section "Line Attributes" on page 161.

2.1.1.3. Text Primitives

A text primitive defines a planar character string.

The text primitives are listed in the section "Text Attributes" on page 163.

2.1.1.4. Marker Primitives

A marker is a symbol at a given position.

The marker primitives are listed in the section "Marker Attributes" on page 162.

2.1.2. Primitive Attributes

A primitive attribute is an output command which in some way affects the appearance of primitives. The primitive attributes are specified alphebetically in *Chapter 8*.

Some primitive attributes apply to any type of primitive, while others apply to only a subset of primitives. The definition of each type of primitive includes a list of the attribute sets (lists of primitive attributes and renderer attributes) which apply to it. *Appendix B* describes each attribute set and the primitives to which the attribute set applies.

2.1.3. Model Identification and Location Aids

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

The Label OC (page 431) marks a location in a structure to assist in structure editing.

These two OCs do not affect PEX rendering.

2.1.4. Application Convenience

The Application Data OC allows applications to store private data. The No-op OC (page 432) allows applications to create place-holder elements. These two OCs do not affect PEX rendering.

2.1.5. Sequential Processing

The Sequential Processing OC (page 433) indicates whether some parallel processing may be applied to subsequent output commands or strict sequential processing is required.

2.2. Renderer

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

A renderer is a PEX resource. It has many attributes which affect its rendering. The **PEXCreateRenderer** (page 531), **PEXChangeRenderer** (page 523), **PEXGetRendererAttributes** (page 537), and **PEXFreeRenderer** (page 536) requests are used to manage a renderer resource.

Request **PEXBeginRendering** (page 520) associates a drawable with a renderer and initializes the renderer. Request **PEXEndRendering** (page 532) completes the rendering and disassociates the renderer and the drawable. Requests **PEXInitRenderer** (page 540), **PEXClearRenderer** (page 524), and **PEXFlushRenderer** (page 534) accomplish subsets of these operations without incurring the overhead of modifying the association between the renderer and the drawable.

Output commands are defined and sent directly to the renderer with request **PEXRenderOutputCommands** for immediate mode rendering. This request must be invoked between a pair of **PEXBeginRendering/PEXEndRendering** requests. Request **PEXRenderNetwork** causes the output commands stored in a structure network to be sent to the renderer. This is referred to as structure mode rendering. Request **PEXRenderNetwork** (page 547) implicitly performs a **PEXBeginRendering/PEXEndRendering** pair.

It is possible to render structure networks even when immediate mode rendering. This is accomplished by using the execute and include structure output commands (see "Structure Hierarchy" on page 829) which, like all output commands, have the same behavior independent of whether the renderer receives them when processing output commands from **PEXRenderOutputCommands** or when traversing a structure.

With request **PEXAccumulateState** (page 519), a structure network's output commands are processed up to a given location such that the renderer's pipeline state is affected, but no primitives are actually rendered. Then request **PEXRenderElements** (page 545) can be used to render a range of output commands within a structure. This provides a quick way to rerender a portion of an image.

2.2.1. Attributes

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

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

The complete list of renderer attributes is described in "Renderer Attributes" on page 509; some of them are reviewed immediately below and throughout the rest of this appendix.

2.2.1.1. Renderer State

One of the renderer's attributes is its state (page 516). When the renderer is not associated with a drawable, its state is *Idle.* The renderer's state is *Rendering* when output primitives are being rendered. There are two remaining modes, *MultipassRendering* and *Picking*.

- Multipass Rendering

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

  If the renderer's state is not *Idle*, the *MultipassRendering* state is initiated with request **PEXInitMultipass** (page 539). All output commands subsequently passed to the renderer are considered part of a pass, many of which may be required to produce the requested effects. The first pass consists of the output commands sent to the renderer between the **PEXInitMultipass** request and a **PEXNextPass** (page 541) or **PEXNextPassWithoutReply** (page 543) request. These latter two requests demarcate the end of the current pass and the beginning of the next. The server expects the same output commands to be sent in each pass.

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

- Picking

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

A point on a primitive is detectable if at the time the primitive is processed the pipeline state's *names* pass the detectability filter (see "Filtering" on page 828) defined by the renderer's *pick inclusion set* and *pick exclusion set* attributes; if the pipeline state's *names* fail the invisibility filter (see "Invisibility" on page 828); the primitive is not culled (as controlled by the pipeline state's *facet culling mode*); and the point on the primitive is not clipped away by the model clipping volume, view clipping planes, the NPC limits, the renderer's *clip list*, or the drawable's boundaries. The geometric definition of the primitive is used, so, for example, the interior of surface primitives with interior style *Hollow* or *Empty* are detectable. It is implementation-dependent whether rendering effects, such as linewidth, which increase the area of the primitive, are considered during picking.

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

A picked primitive is returned to the client in the form of a pick path. The pick path shows each level of hierarchy from the first structure to the picked primitive. Each member of the pick path consists of a structure resource identifier, offset, and pick identifier. The offset is the element position of the element invoking the next structure in the hierarchy or the element position of the picked primitive. The pick path is formed from the renderer's *current path* (see "Current Path" on page 823) attribute and the pipeline state's *pick identifier.*

Structure mode clients wishing to pick just one primitive use request **PEXPickOne**. Immediate mode clients use request **PEXBeginPickOne** (page 565), followed by the **PEXRenderOutputCommands** requests, and finish with request **PEXEndPickOne** (page 569)**.**

Structure mode clients wishing to pick all eligible primitives initialize the renderer's *pick start path* attribute indicating the location in a structure network at which to start the pick. Request **PEXPickAll** (page 571) is then invoked. If **PEXPickAll** cannot return all the pick paths, it returns as many as it can and indicates more pick paths are available. The request also sets the *pick start path* renderer attribute to reflect the point at which it should continue the pick. The client continues invoking **PEXPickAll**, fetching pick paths until **PEXPickAll** indicates there are no more paths.

The mechanism differs slightly for immediate mode clients which wish to pick all eligible primitives. These clients invoke **PEXBeginPickAll**. PEX begins searching for hits as soon as **PEXRenderOutputCommands** is invoked. The client ends the pick by invoking request **PEXEndPickAll** (page 567). If **PEXEndPickAll** cannot return all the pick paths, it returns as many as it can and indicates more paths are available. The client then uses the renderer's *pick start path* attribute to determine which output commands it should send to allow PEX to resume the pick when **PEXBeginPickAll** is next invoked. Event *MaxHitsReached* is generated when the server has determined as many pick paths as it can. The client can take advantage of this event to avoid sending output commands which will have to be resent in subsequent **PEXBeginPickAll/PEXEndPickAll** rounds.

2.2.1.2. Current Path

The renderer's *current path* attribute consists of structure resource identifier and offset pairs. Initially the path has a single member with both structure identifier and offset set to zero. Each time a structure is executed or included, or a **PEXBeginStructure** or **PEXRenderNetwork** request is received, a new pair is added to the current path, with the given structure identifier and an initial offset of zero. Whenever processing of an executed or included structure is completed, or a **PEXEndStructure** request is received, the last pair is removed from the current path. As each output command is processed, the offset of the last member of the current path is incremented.

2.2.1.3. Echo Mode

The *echo mode* renderer attribute controls whether primitives, as they are rendered, should be unechoed, echoed, or drawn normally. An echoed primitive's visual attributes are modified to emphasize the primitive. When echoing is enabled, the *echo method* (page 512) and *echo color* (page 512) renderer attributes indicate how the echoing is to be accomplished. If a primitive is designated to be both highlighted and echoed with color methods, it is echoed and not highlighted.

2.2.1.4. Transparency

The transparency of primitives is controlled by the *transparency method* renderer attribute. In addition to *None*, there are four transparency methods: *ScreenDoor*, *AlphaBlend*, *TwoPass*, and *Multipass*. The alpha component of each primitive's color is used in the transparency computations. An alpha value of 0.0 is transparent (i.e., zero coverage), and an alpha value of 1.0 is opaque (i.e., 100% coverage). If the primitive's color does not include an alpha component, a default alpha value is used. For surface primitives, the default alpha value is (1.0 - *transmission_coefficient*). The back-facing transmission coefficient is used when rendering a back-facing facet when the pipeline state's *facet distinguish flag* is *True*. The transmission coefficients are selected from the surface reflection attributes (and back-facing surface reflection attributes) ("Lighting" on page 833). For all other primitives, the default alpha value is 1.0.

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

The pipeline state's *alpha blend function*, modified with the Alpha Blend Function OC (page 265), allows control over the alpha blending function used when the *AlphaBlend* transparency method is used. Some of the alpha blend functions require an alpha buffer ("The Alpha Buffer" on page 826), an ancillary buffer to the drawable.

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

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

2.2.2. Pipeline State

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

The renderer actually maintains a stack of pipeline states. Each time an Execute Structure OC (page 428), a Conditional Execute Structure OC (page 425), or a **PEXBeginStructure** request initiates the traversal of a new structure, a copy of the pipeline state's attributes are saved. Its *global transform* and *global transform hints* fields are immediately set to the current composite modeling transformation ("Model, Text Local, and World Coordinate Spaces" on page 832), and the *local transform* and *local transform hints* fields are immediately set to the identity matrix. Traversal of the new structure potentially causes any of the pipeline state fields to be further modified. When the execution of the new structure is terminated, because its last OC was processed, or a Conditional Return OC (page 427) or **PEXEndStructure** request forced its completion, the most recently saved copy of the pipeline state is restored. This mechanism is used to create hierarchical models.

2.2.2.1. Pipeline Context

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

The **PEXCreatePipelineContext**, **PEXChangePipelineContext**, **PEXCopyPipelineContext**, **PEXGetPipelineContext**, and **PEXFreePipelineContext** requests are used to manage a pipeline context resource.

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

2.3. Drawables and their Ancillary Buffers

The renderer draws 3D geometrical objects into the X drawable bound to it. The type (e.g., window, pixmap, or MBX buffer) and visual identifier (if applicable) of those drawables supported by PEX for a given screen, depth, and visual are returned by request **PEXMatchRendererTargets** (page 120).

It is recommended the Multi-buffering Extension to X (MBX) be used for double-buffering. The MBX image buffers are alternately bound to a renderer to achieve this effect.

The Extended Visual Extension to X (EVE) is recommended for inquiring the drawable's frame buffer capabilities, including which ancillary buffers are supported. PEX does not provide such inquiry support. PEX uses the accumulation, alpha, and Z-buffer ancillary buffers as described below.

2.3.1. HLHSR and the Z-buffer

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

HLHSR is controlled by the *HLHSR mode* renderer attribute (page 513). HLHSR is disabled with mode *Off*. Mode *ZBuffer* enables HLHSR using the Z-buffer. Mode *ZBufferId* also selects the Z-buffer, but indicates its usage is further controlled by the pipeline state's *HLHSR identifier*, modified with the HLHSR Identifier OC (page 323). This allows HLHSR to be specified as a part of the graphical model. The HLHSR Identifier OC specifies three usages of the Z-buffer: *Disable*, *Enable*, and *EnableTestOnly*. Usage *EnableTestOnly* specifies a pixel is, as usual, compared against the Z-buffer to determine whether the pixel is rendered, but if it is determined the pixel should be rendered, the Z-buffer is not updated with the pixel's Z-value. This capability is typically used to temporarily add objects to the image, which respect depth sorting, but which are later removed by rerendering the affected area of the image.

The pipeline state's *Z-buffer test*, modified with the Z-buffer Test OC (page 419), provides still further control over the usage of the Z-buffer. The Z-buffer test allows the client to choose which results of the comparison between the pixel's depth value and the Z-buffer's value result in a determination the pixel should be rendered. It is not specified in exactly which coordinate space HLHSR is accomplished, but it is guaranteed it is done in a right-handed coordinate system where a larger Z-value is closer to the eyepoint. Therefore typically the selected Z-buffer test is *GreaterThan* or *GreaterThanOrEqual*. However the Z-buffer Test OC allows any relational test to be selected. Additionally it supports *Pass*, which by causing all pixels to be rendered, in affect disables depth sorting but ensures the Z-buffer is still updated if HLHSR is enabled. The Z-buffer test is in effect whenever any HLHSR method using the Z-buffer is enabled.

Whenever an implicit or explicit **PEXBeginRendering** occurs and either the *HLHSR mode* renderer attribute selects an HLHSR method requiring a Z-buffer, or the *clear Z* renderer attribute (page 511) is *True*, a Z-buffer is allocated, if necessary, and bound to the drawable. (A new Z-buffer is allocated only if a suitable one is not already bound to the drawable. Sometimes a drawable will have an unsuitable Z-buffer, e.g., if a resize occurred and the Z-buffer has not yet been resized.) If these conditions do not occur but the *HLHSR mode* renderer attribute is later changed with request **PEXChangeRenderer** to indicate a Z-buffer is required, the Z-buffer is allocated, if necessary, at that time. Once allocated, the Z-buffer remains bound to the drawable until the drawable is freed. As mentioned above, the buffer is reallocated as necessary.

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

The *clear Z value* renderer attribute (page 511) specifies the initial Z-buffer value. All Z-buffer values within the renderer's clip list (set by the *clip list* renderer attribute (page 512)) are set to this value, if the *clear Z* renderer attribute is *True*, when a Z-buffer is allocated or whenever an implicit or explicit **PEXBeginRendering** occurs and a Z-buffer is bound to the drawable. The Z-buffer can also be cleared by invoking request **PEXClearRenderer** (page 524) with the *clear_Z_buffer* parameter set.

The client can save and restore a drawable's Z-buffer on the server with requests **PEXCopyZBufferToPixmap** and **PEXCopyPixmapToZBuffer** , which use the implementation-dependent format and range of the Z-buffer values. The client can inquire and modify any arbitrary rectangle of the Z-buffer with requests **PEXGetZBuffer** and **PEXPutZBuffer**, which represent the Z-buffer values as floating point values in the range 0.0 to 1.0.

2.3.2. The Alpha Buffer

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

Whenever an implicit or explicit **PEXBeginRendering** occurs and either the *transparency method* renderer attribute selects a transparency method requiring an alpha buffer, or the *clear alpha* renderer attribute (page 511) is *True*, an alpha buffer is allocated, if necessary, and bound to the drawable. (A new alpha buffer is allocated only if a suitable one is not already bound to the drawable. Sometimes a drawable will have an unsuitable alpha buffer, e.g., if a resize occurred and the alpha buffer has not yet been resized.) If these conditions do not occur but the *transparency method* renderer attribute is later changed with request **PEXChangeRenderer** to indicate an alpha buffer is required, the alpha buffer is allocated, if necessary, at that time. Note transparency method *AlphaBlend* does not cause an alpha buffer to be allocated unless the pipeline state's *alpha blend function* also selects a function requiring an alpha buffer. Therefore a change in the pipeline state's *alpha blend function* can also trigger the allocation of the alpha buffer. Once allocated, the alpha buffer remains bound to the drawable until the drawable is freed. As mentioned above, the buffer is reallocated as necessary.

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

The alpha component of the *background color* renderer attribute (page 511) specifies the initial alpha buffer value. If the background color does not have an alpha component, a default value of 1.0 (opaque) is used. All alpha buffer values within the renderer's clip list (set by the *clip list* renderer attribute (page 512)) are set to this value, if the *clear alpha* renderer attribute is *True*, when an alpha buffer is allocated or whenever an implicit or explicit **PEXBeginRendering** occurs and an alpha buffer is bound to the drawable. The alpha buffer can also be cleared by invoking request **PEXClearRenderer** (page 524) with the *clear_alpha_buffer* parameter set.

The client can save and restore a drawable's alpha buffer on the server with requests **PEXCopyAlphaBufferToPixmap** and **PEXCopyPixmapToAlphaBuffer**.

2.3.3. The Accumulation Buffer

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

Color or *TrueColor*. The buffer must be at least as deep as the drawable.

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

2.4. Lookup Tables

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

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

The **PEXCreateLookupTable** (page 476), **PEXCopyLookupTable**, and **PEXFreeLookupTable** requests are used to manage a lookup table resource. Individual entries of a lookup table are set and inquired with the **PEXChangeTableValues** (page 474) and **PEXGetTableValues** (page 482) requests. All of an entry's values, or just a subset of the values, may be set or inquired with these requests.

Request **PEXGetTableInfo** (page 481) is used to determine the total number of entries which can be defined for a type of lookup table and which entries, if any, are automatically defined upon table creation. The automatically defined entries, known as the predefined entries, are a convenience and may be modified by the client. The server may support up to 65,535 entries in a table. Even in a table which supports fewer entries, indices in the range 0 - 65,534, or 1 - 65,535, depending on the type of table, are valid. That is, the entries do not have to be consecutively indexed. Request **PEXGetDefinedIndices** (page 480) returns the list of entries defined for a given table. An entry is defined either because it is a predefined entry or because request **PEXChangeTableValues** was invoked with its index. A defined entry is undefined with request **PEXDeleteTableEntries** (page 477).

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

2.4.1. Bundle Tables

Some of the lookup tables, referred to as bundle tables, provide an alternate means of setting attributes which are also settable through an OC. Each of these tables define a set of related line, surface, surface edge, data mapping, marker, or text attributes. However values within a bundle table are individually selected. Which source is used during rendering is controlled with the attribute's aspect source flag (ASF). If the ASF is *Individual*, the pipeline state is the source of the attribute's value. If the ASF is *Bundled*, an entry of the attribute's corresponding bundle table, selected by the bundle table's index in the pipeline state, is the source of the attribute's value. The aspect source flags themselves are part of the pipeline state, modified by the Set Individual ASF OC (page 325).

The interior bundle table, which contains surface attributes, is unique in that there are two modes of accessing it. By default, the pipeline state's *interior bundle index* selects an entry which is used for all of the bundled surface attributes. However the *PHIGS interior bundle mode* renderer attribute (page 515) may be used to select a different access method which matches the PHIGS PLUS surface attribute bundling. When the PHIGS interior bundle mode is *True*, five different pipeline state indices, the *interior bundle index*, *back-facing interior bundle index*, *reflectance bundle index*, *back-facing reflectance bundle index*, and *parametric surface bundle index*, are used to separately select subsets of five different entries.

2.5. Name Sets

With the pipeline state's *names* field, primitives can be labeled for the purposes of highlighting, enabling and disabling their rendering, and picking. Each name is simply an integer. Names are added and removed from the graphics model with the Add Names to Name Set (page 262) and Remove Names from Name Set (page 263) OCs.

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

2.5.1. Filtering

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

2.5.2. Highlighting

A primitive is highlighted when the pipeline state's *names* pass the highlighting filter defined by the renderer's *highlight inclusion set* and *highlight exclusion set* attributes. When a primitive is highlighted, its visual attributes are modified to emphasize it. Control over the attribute modification is provided through the highlight lookup table. Each entry of this table sets a highlight method. The entry of the highlight table used to highlight a primitive is selected by the pipeline state's *highlight index*, modified with the Highlight Index OC (page 322). The only way to unhighlight a primitive is to clear and rerender.

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

2.5.3. Invisibility

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

2.5.4. Detectability

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

2.6. Structures

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

OCs are added to a structure with request **PEXStoreElements**. Elements are copied from one structure to another with request **PEXCopyElements**. Elements are inquired with requests **PEXGetElementInfo** and **PEXFetchElements**. Elements are deleted with requests **PEXDeleteElements**, **PEXDeleteElementsBetweenLabels**, and **PEXDeleteElementsToLabel**.

2.6.1. Structure Hierarchy

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

The Execute Structure (page 428) and Include Structure (page 429) OCs invoke another structure. That is, when one of these output commands is processed, processing continues with the output commands stored in the invoked structure. Execute Structure causes the pipeline state to be saved and a new copy to be made and modified ("Pipeline State" on page 824.) The new copy is deleted and the saved copy restored when processing of the invoked structure is complete. In the case of Include Structure, there is no save and restore of the pipeline state. That is, modifications to the pipeline state by the processing of the invoked structure carry through when processing of the invoking structure is later continued. The execution and inclusion of structures affect the renderer's *current path* as described in "Current Path" on page 823.

There are also a set of output commands which allow structures to be conditionally invoked and a conditional early return from an invoked structure. The Condition Flags (page 424), Test Extent (page 434), and Test Extent 2D (page 436) OCs are used to modify the *condition flags* field of the pipeline state. The extent is a bounding box which is transformed to device coordinates. A clip test and a size test are conducted against the extent, and the results are used to modify the *condition flags*. The clip test is typically used to conditionally skip the processing of primitives which will be clipped. The size test is a comparison between the extent's DC space diagonal length and an extent size selected from an extent size lookup table (page 448). The size test can be used to conditionally render different OCs representing different versions of the model based on the screen size of the model. The Conditional Execute Structure (page 425), Conditional Include Structure (page 429), and Conditional Return (page 427) OCs use the condition flags to affect the flow of output commands processed.

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

All structures referencing a structure, called the parents of that structure, plus all their parents and so on up to root structures, constitute the ancestors of the given structure. **PEXGetAncestors** (page 596) returns a list of paths down to a specific structure. Similarly, all structures referenced by a given structure are called its children, and the children of a structure together with all their children and so on down to structures containing no structure references OCs are called its descendents. **PEXGetDescendents** (page 598) returns a list of the reference path starting from a given structure, and **PEXGetStructuresInNetwork** (page 605) returns a single list of all descendents.

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

2.6.2. Structure Attributes

The *element pointer* structure attribute indicates a position within the structure. Each element within the structure is identified by its position. Position 1 is the first element in the structure. Elements beyond the first are assigned sequential positions. Position 0 corresponds to the location before the first element in the structure. An element's position changes when elements between it and position 0 are added or removed. The element pointer can be positioned to any element in the structure or to position 0 with one of the following requests: **PEXSetElementPointer**, **PEXSetElementPointerAtLabel**, or **PEXSetElementPointerAtPickID** (page 609). The element pointer is also changed when elements are added or removed. The requests that add and remove elements define their effect on the element pointer. A structure's element pointer is initially 0.

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

The *permission* structure attribute indicates the extent to which a server can reorganize the structure's data. If the permission is *ReadWrite*, the elements and their contents are maintained exactly as they were created. If the permission is *WriteOnly*, the server may reorganize the data of individual elements. If the permission is *Locked*, the server may also reorganize the data represented by groups of elements. Reorganization results in restrictions on the operations allowed on a structure but does not alter the resulting image when the structure is rendered. A structure's permission does not affect the permission of the structures it references. Also pick paths through structures whose permission is not *ReadWrite* reflect accurate offsets, but the offset of the picked primitive itself is only guaranteed to be accurate if the permission of its structure is *ReadWrite*. A structure's permission is modified with request **PEXSetStructurePermission**.

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

2.7. Fonts

A PEX font is a resource representing a text font supported by PEX. It is analogous to the X font resource, which represents text fonts supported by the X server. It is possible for the same font to be supported by both or just one of X and PEX. Requests **PEXListFonts** (page 712) and **PEXListFontsWithInfo** (page 713) indicate which fonts are supported by PEX.

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

Each PEX font has the following information defined for it: the first, last, and default glyphs, whether all the glyphs exist, the font's properties, and whether the font can be rendered with PEX's *StrokePrecision* text precision (page 410). This information can be returned for a particular PEX font resource with request **PEXQueryFont** (page 717). Request **PEXListFontsWithInfo** returns this information for each font supported by PEX.

PEX fonts to be potentially applied to a single string are defined as a font group. Each font group corresponds to an entry of the text font table (page 459). The pipeline state's *text font index*, modified with the Text Font Index OC (page 407), selects an entry of the text font table currently bound to the renderer. The fields in this entry are applied to each text and annotation text primitive. Each of these primitives defines a single string which is a list of mono-encodings. The rendered string is formed from its mono-encodings in the order the mono-encodings are given. There is no space between the last character of a mono-encoding and the first character of the next mono-encoding beyond the spacing applied to any character in the string. For each mono-encoding, the character width is specified (8-, 16-, or 32-bit), and a font is selected from the font group selected by the pipeline state's *text font index*.

Request **PEXQueryTextExtents** returns the text local coordinate space ("Model, Text Local, and World Coordinate Spaces" on page 832) extent (bounding box) and connection point for a text string with a given set of text attributes applied.

2.8. Search Context

The search context is a PEX resource which performs a view-independent pick on a geometrical object defined by a structure network. The pick is done with a structure network only; it is completely independent of a renderer resource. The search context resource is managed with requests **PEXCreateSearchContext**, **PEXChangeSearchContext**, **PEXCopySearchContext**, **PEXGetSearchContext** (page 666), and **PEXFreeSearchContext** (page 665).

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

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

Request **PEXSearchNetwork** (page 667) conducts the pick. The first primitive which meets the search context criteria, if one exists, is returned.

2.9. PHIGS Workstation and Pick Measure

The PHIGS workstation (*Chapter 19*) and pick measure (*Chapter 20*) PEX resources provide functionality that is backwards compatible with PEX 5.1. They have not been updated in PEX 5.2.

2.10. Enumerated Types and Implementation-dependent Constants

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

Enumerated types provide a list of supported values for a given functionality. Each type consists of its class, which names the functionality, and a list of values. The values are represented by a numerical label and a mnemonic label, which is an ISO 8859/1 string describing the value. Each enumerated type is defined with related functionality throughout the specification. They are always given in a table labeled "Enumerated Type <*class*>". The table lists the numerical label, mnemonic, and a description of each value. Request **PEXGetEnumeratedTypeInfo** (page 99) is used to query the supported values for a list of enumerated types. It can optionally return the numerical label and/or mnemonic of the supported values for each inquired enumerated class.

PEX uses implementation-dependent constants to indicate whether a specific optional functionality is supported (using constants with *True/False* values) and to indicate implementation-dependent non-negative integer or floating-point limits. Request **PEXGetImpDepConstants** (page 111) is used to query the values of a list of implementation-dependent constants. The constants are defined with this request but are referenced by related functionality throughout the specification.

PEX Rendering

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

2.11. Transformation Pipeline

2.11.1. Model, Text Local, and World Coordinate Spaces

Primitives are defined in their own model coordinate (MC) space. This allows primitives to be specified in whatever space is convenient. To place the primitives in a common space, they are each transformed by the composite modeling transformation into world coordinate (WC) space. The *local transform* and *local transform hints* fields of the pipeline state, which define the local modeling transformation, are modified with the Local Transform (page 348) and Local Transform 2D (page 350) OCs. The *global transform* and *global transform hints* fields of the pipeline state, which define the global modeling transformation, are modified with the Global Transform (page 316) and Global Transform 2D (page 320) OCs. These transforms are used to create the composite modeling transformation. Model coordinate space, the composite modeling transformation, and world coordinate space are defined in detail by PHIGS 4.7.2.

Each Text (page 233) and Text 2D (page 235) primitive defines its own text local coordinate space. This space is an orientation of model coordinate space with the same scaling as MC space. The definition of the text primitives and many of the text attributes make use of the text local coordinate space. Each Annotation Text (page 174) and Annotation Text 2D (page 176) primitive also defines its own text local coordinate space. In this case, the text local coordinate space also has the same scaling as MC space but is specified in NPC space.

2.11.2. Model Clipping

Model clipping allows the client to specify objects outside a volume defined in model coordinates which are not to be rendered. The pipeline state's *model clip indicator*, which indicates whether model clipping is enabled, is modified with the Model Clipping OC (page 359). The Model Clipping Volume (page 360) and Model Clipping Volume 2D (page 362) OCs incrementally modify the *model clip volume* field of the pipeline state. The Restore Model Clipping Volume OC (page 264) restores the pipeline state's *model clip volume* to the value inherited by the structure being traversed. The model clip volume is transformed by the composite modeling transformation to world coordinates where the clip actually takes place. Model clipping is defined in detail by PHIGS 4.7.2.

2.11.3. Viewing

The pipeline state's *view index*, modified by the View Index OC (page 418), selects an entry of the view table (page 472). The viewing process transforms a volume of world coordinate space into a volume in normalized projection coordinate (NPC) space. Multiple views may be applied within a single image. All the views are oriented relative to each other in all three dimensions in NPC space.

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

Viewing is defined in detail by PHIGS 4.7.4 and 4.7.5.

2.11.4. NPC Space to Device Coordinate Space

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

The *NPC subvolume* renderer attribute (page 515) defines which portion of NPC space will be mapped to the drawable. The *viewport* renderer attribute (page 518) defines the portion of the drawable to which this mapping will occur. The actual viewport is adjusted so the aspect ratios for *x*, *y*, and *z* are the same in DC space as in NPC space.That is, the actual viewport is the largest parallelepiped that can fit into the defined viewport and maintain the *x* and *y* aspect ratios while mapping the *z* extent of the NPC subvolume to the entire *z* extent of the defined viewport. This means there is uniform scaling in *x* and *y* from NPC to DC space, but there may be some unused rendering space at the top right side of the defined viewport.

If the viewport's boundaries extend outside the drawable, rendering to the viewport is clipped by the drawable. Additionally, rendering to the viewport is clipped in the *x* and *y* directions as specified by the *clip list* renderer attribute (page 512). The clip list is defined by a list of non-overlapping rectangles defined in DC space.

2.12. Rendering Pipeline

2.12.1. Color

PEX colors are defined by a type and list of components. Type *Indexed* indicates at the time of rendering the color is to be taken from the color lookup table (page 441) bound to the renderer. The list of components for type *Indexed* consists of a single value, an index into the color lookup table. Each entry of the color lookup table contains the definition of one color. The other color types are all color models, such as red-green-blue (RGB). The list of registered PEX color types includes a definition and valid value range for each of the components of each color type.

As a primitive enters the rendering pipeline, its intrinsic color is determined. For the surface primitives optionally including color data, this is done through data mapping. The purpose of data mapping is to specify whether the color optionally defined for each of a primitive's vertices, optionally defined for each of the primitive's facets, or the pipeline state's *surface color* or *bf surface color* is used to determine the surface primitive's intrinsic color. The pipeline state surface and back-facing surface colors, also known as "aspect colors", are modified by the Surface Color (page 390), Surface Color Index (page 392), Back-facing Surface Color (page 294), and Back-facing Surface Color Index (page 296) OCs. Data mapping method *None* sets the priority to vertex color (if present), overrides facet color (if present), overrides aspect color. Method *Color* allows the client to prioritize this list of sources itself. The data mapping method applied is selected from either the pipeline state or data mapping lookup table (page 445) according to the pipeline state's *data map method ASF*. If the ASF is *Individual*, the pipeline state's *data map method*, modified with the Data Mapping Method OC (page 308), is used. Otherwise the ASF is *Bundled* and the pipeline state's *data map index*, modified with the Data Mapping Index OC (page 307), selects an entry of the data mapping lookup table. If face distinguishing is enabled and the surface to be data mapped is back-facing, the pipeline state's *bf data map method ASF, bf data map method* (modified with the Back-facing Data Mapping Method OC (page 308)), and *bf data map index* (modified with the Back-facing Data Mapping Index OC(page 307)) are used similarly.

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

The PEX implementation actually performs rendering in the pipeline state's *rendering color model*, modified by the Rendering Color Model OC (page 383). All colors entering the rendering pipeline, such as a primitive's intrinsic color, are transformed into the rendering color model. All rendering calculations are then done in the rendering color model.

2.12.2. Lighting

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

The light sources are defined in the light lookup table (page 452). There are four types of light sources. Each type has a color. Additional data is needed to define the light source depending on its type. The *Ambient* light source type illuminates primitives uniformly independent of location and orientation. The *WCS_Vector* light source type simulates light emanating from infinity along a particular direction. The *WCS_Point* light source type simulates light emanating from all directions from a particular position. The *WCS_Spot* light source type simulates light emanating from a particular position within a cone of influence centered along a particular direction. Lights are enabled by placing their lookup table indices in the pipeline state's *light source state*, modified with the Light Source State OC (page 335).

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

$$L_a = \frac{1}{C_1 + C_2|O_p - L_p| + C_3|O_p - L_p|^2}$$

The extent of the reflectance calculation is controlled by the reflection model. If the model is *None*, no calculation is performed. Otherwise, for each enabled light source, model *Ambient* indicates only the ambient terms of the reflectance calculation are used, model *Diffuse* indicates the ambient and diffuse terms are used, and *Specular* indicates all the terms, ambient, diffuse, and specular, are used. The reflection model applied is selected from either the pipeline state or interior bundle table according to the pipeline state's *reflection model ASF*. If the ASF is *Individual*, the pipeline state's *reflection model*, modified with the Surface Reflection Model OC (page 381), is used. Otherwise the ASF is *Bundled* and the pipeline state's *interior index*, modified with the Interior Bundle Index OC (page 378), selects an entry of the interior bundle table from which the *surface reflection model* field is used. If face distinguishing is enabled and the surface to be lit is back-facing, the pipeline state's *bf reflection model ASF* and *bf reflection model*, and the *back-facing surface reflection model* field of the interior bundle table are used similarly. (Selection of a bundled interior attribute is controlled differently if the renderer's *PHIGS interior bundle mode* attribute is set.)

The factors used in the reflectance calculation are defined by the lit point's position, reflectance normal, and intrinsic color; the light source's definition; the eyepoint ("Viewing" on page 832); and the reflection attributes. A facet's reflectance normal defines the orientation of a facet at the point at which a reflectance calculation is to be performed. See PHIGS PLUS 4.5.5.3 for a definition of how the reflectance normal is determined for each front-facing and back-facing facet. The back-facing reflectance normal is used for all back-facing facets, independent of the face distinguishing mode.

The reflection attributes define the ambient, diffuse, and specular coefficients, and the specular color and concentration applied to the primitive. Although the transmission coefficient is included in the reflection attributes, it is used for alpha blending rather than by the reflectance calculation. The reflection attributes are like the reflection model in that they are selected from either the pipeline state or interior bundle table according to the pipeline state's *reflection attributes ASF*. Either the pipeline state's *reflection attributes*, modified with the Surface Reflection Attributes OC (page 379), or the *surface reflection attributes* field of the interior bundle table is the source of the reflection attributes. If face distinguishing is enabled and the surface to be lit is back-facing, the pipeline state's *bf reflection attributes ASF* and *bf reflection attributes*, and the *back-facing surface reflection attributes* field of the interior bundle table are used similarly.

2.12.3. Shading

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

Shading of linear primitives is controlled by the polyline interpolation method. There are two methods: *None*, which disables interpolation; and *Color*, which enables interpolation of color between vertices of the vertex colors (if they are present). The pipeline state's *polyline interpolation method ASF* determines whether the source of the polyline interpolation method is the pipeline state's *polyline interpolation method*, modified by the Polyline Interpolation Method OC (page 374), or the *polyline interpolation method* field of the line bundle table (page 454) entry selected by the pipeline state's *line bundle index* (modified by the Line Bundle Index OC (page 337)).

Shading of surface primitives is controlled by the surface interpolation method. The surface interpolation method is selected similarly to the polyline interpolation method except the pipeline state's *surface interpolation method ASF*, *surface interpolation method* (modified by the Surface Interpolation Method OC (page 399)), and *interior bundle index,* and the *surface interpolation method* field of the interior bundle table entry are used. If face distinguishing is enabled and the surface to be shaded is back-facing, the pipeline state's *bf surface interpolation method ASF* and *bf surface interpolation method* (modified by the Back-facing Surface Interpolation Method OC (page 297)), and the *back-facing surface interpolation method* field of the interior bundle table entry are used.

The surface interpolation methods are described below. They are listed in order of increasing accuracy at the cost of increasing computation. Note their behavior is dependent on the interior style, which is selected similarly to the surface interpolation method except the pipeline state's *interior style ASF* and *interior style* (modified by the Surface Interior Style OC (page 331)), and the *surface interior style* field of the interior bundle table's entry are used. In the back-facing case, the pipeline state's *bf interior style ASF* and *bf interior style* (modified by the Back-facing Surface Interior Style OC (page 283)), and the *back-facing surface interior style* interior bundle table field are used.

- Surface Interpolation Method *None*

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

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

- Surface Interpolation Method *Color*

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

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

- Surface Interpolation Method *DotProduct*

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

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

- Surface Interpolation Method *Normal*

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

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

2.12.4. Texture Mapping

Texture mapping simulates material properties for surface primitives by adding visual detail with color. This is accomplished by blending the sampling of a map of colors with the primitive's color.

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

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

A texture map resource is enabled by referencing it from an entry of the texture mapping lookup table (page 460) and enabling that entry with the pipeline state's *active textures*, modified with the Surface Active Textures OC (page 385). The corresponding *bf active textures* pipeline state field, modified with the Back-facing Surface Active Textures OC (page 292), is applied to back-facing surfaces when face distinguishing is enabled. Multiple texture maps may be selected for application.

2.12.4.1. The Texture Mapping Lookup Table

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

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

Several of the fields control parameterization, that is, how texels are mapped to a facet. The *parameterization method* field (page 462) specifies how the texture map coordinates for each vertex are determined. The texture map coordinates select a position in the texture map. The portion of a texture map selected by the vertices's texture map coordinates is then applied to the facet. The *VertexCoordMC, VertexCoordWC, VertexNormalMC, VertexNormalWC, NURBUV,* and *FloatData* parameterization methods use the vertex, normal, u-v parameterization, or floating-point data list data directly as the texture map coordinates. Field *fp data index* of the texture mapping lookup table defines the index within the floating-point data list of the first of the consecutive texture mapping coordinates when the parameterization method is *FloatData.* The *ReflectionSphereVRC* and *ReflectionSphereWC* parameterization methods select texture map coordinates by picking them off a conceptual sphere surrounding the facet. A special reflection vector calculated for each vertex is used to select a point on the sphere. The resulting texture mapping is often called environment mapping since the texturing is view dependent - the facet appears to reflect fixed objects as the facet is moved around. The *reflection matrix and hints* field of the texture mapping lookup table is applied only when one of these environment mapping parameterization methods is selected. The reflection matrix is used to transform the reflection vectors before they select the texture coordinates from the sphere, which has a fixed orientation.

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

Another group of fields in the texture mapping lookup table control the sampling of the color mipmap. When one pixel corresponds to multiple texels, the *min sample method* field (page 469) indicates how the texel data is sampled down to derive a single color value. The *mag sample method* field controls the opposite case of one pixel corresponding to less than a single texel. The sample methods support a wide range of sampling options to determine the color value, from *SingleBase*, which uses a single texel in the base level of the texture map, to *WeightedBetweenMipmaps*, where the weighted averages of 2^n texels on each of the two levels nearest the sample depth are determined, then interpolated to the sample depth. The *sampling hints* field (page 471) potentially adjusts the sampling depth, making the texturing more detailed or blurred than it otherwise would have been. It is also used to indicate blurring is unacceptably high due to low spatial frequency in a particular direction in the color mipmap.

Textures sometimes map to something less than the entire facet. The *t0, t1,* and *t2 boundary condition* fields specify how the region in each direction outside the mapped texture is handled. Condition *Wrap* indicates the texture is repeated across the facet. Condition *Mirror* also indicates the texture is repeated, but alternately in reverse order. Condition *ClampBorder* indicates the closest border texel is used to determine the sampled color value beyond that border. Condition *ClampExplicit* indicates the *boundary clamp method* field is used to determine the boundary condition. (This level of indirection exists to prevent ambiguity in regions which extend outside the mapped texture in more than one direction.) The *Discontinue* boundary clamp method indicates texturing is to be discontinued beyond the mapped texture, that is, the primitive's color before applying that texture map is preserved. Boundary clamp method *Color* indicates the *boundary clamp color* field specifies the texture's color value. For the sampling methods that use multiple texels on a mipmap level to compute a weighted average, it is possible that the neighboring texels do not exist because the implementation is sampling near the texture map boundary. In these cases, the implementation synthesizes the missing texels by using the boundary condition in effect. However, a boundary condition of *ClampExplicit* with a clamping method of *Discontinue* does not cause the primitive's intrinsic color to be included in the weighted average.

The remaining fields in the texture mapping lookup table specify the blending of the color values derived from the texture map and boundary conditions with the primitive's color. The *composition method* field (page 465) specifies how this blending is done. Method *Replace* indicates the primitive's color is replaced with the derived texture color. The alpha component is set to 1.0 if the texture map didn't include an alpha component. Method *Modulate* specifies the primitive's color and derived texture color are multiplied. Method *BlendEnvColor* indicates the derived texture color is multiplied by the color specified in the *environment color* field. The result is added to the multiplication of the primitive's color and the inverse of the derived texture color. The primitive's alpha component is just multiplied by the derived texture alpha component (1.0 if not present). Method *Decal* specifies the derived texture color is multiplied by its alpha component. The result is added to the multiplication of the primitive's color and the inverse of the derived texture alpha component. The primitive's alpha component is not modified. Method *DecalBackground* is identical to method *Decal* except the color specified in the *background color* field is used instead of the primitive's color.

2.12.4.2. Texture Mapping Attribute OCs

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

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

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

The *tm resource hints* field of the pipeline state, modified by the Texture Mapping Resource Hints OC (page 413), allow the client to suggest how the server could most efficiently manage the texture map resources. The client expresses a preference for whether *Speed* or *Space* is more critical, or whether it doesn't matter. Also the client can provide a prioritized usage list of texture mapping table entries.

2.12.5. Depth Cueing

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

During rendering, an entry of the depth cueing lookup table (page 446) is always selected by the pipeline state's *depth cue index*, modified by the Depth Cue Index OC (page 310). Depth cueing is actually enabled or disabled by the *depth cue mode* field of the selected depth cueing lookup table. If enabled, the *front* and *back reference plane* fields define constant *Z*-value planes in NPC space. Typically the color in the *depth cue color* field is blended with the primitive's color in front of the front reference plane according to the multiplicative factor defined in the *front scale factor* field. The *back scale factor* field defines the factor used for the blend behind the back reference plane. These factors are interpolated between the planes to determine the blending in that region. Suggested depth cueing formulae are given in Annex E of PHIGS PLUS.

2.12.6. Data Mapping / Lighting / Shading / Texture Mapping / Depth Cueing Diagrams

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

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

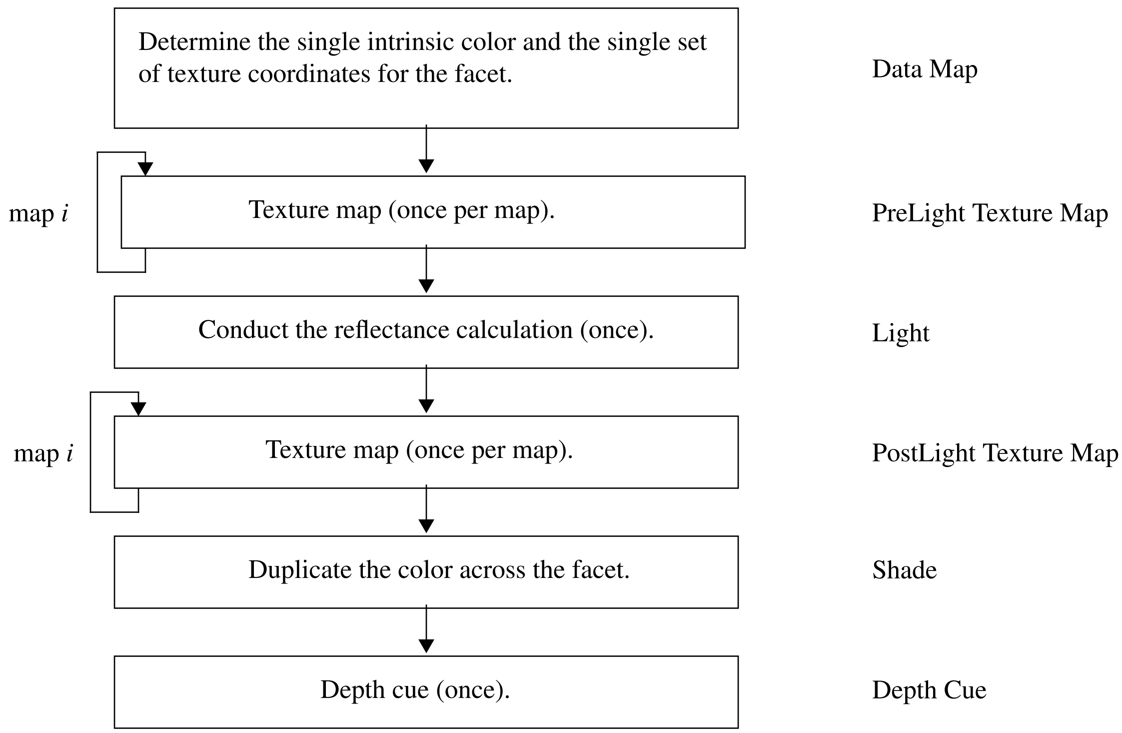


Figure 2.***Surface Interpolation Method None and Texture Mapping Sample Frequency Pixel***

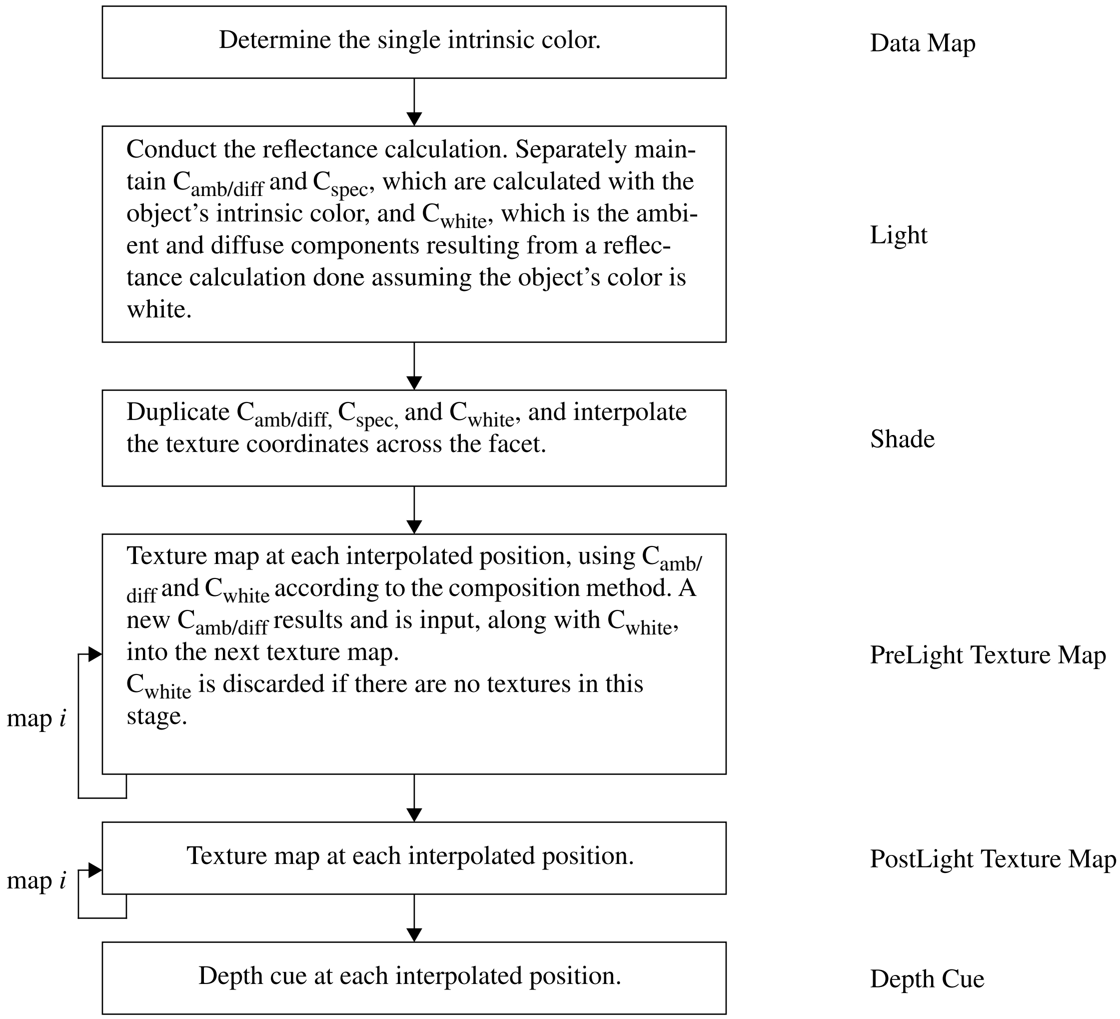


Figure 3. ***Surface Interpolation Method Color and Texture Mapping Sample Frequency InterpDep (per vertex)***

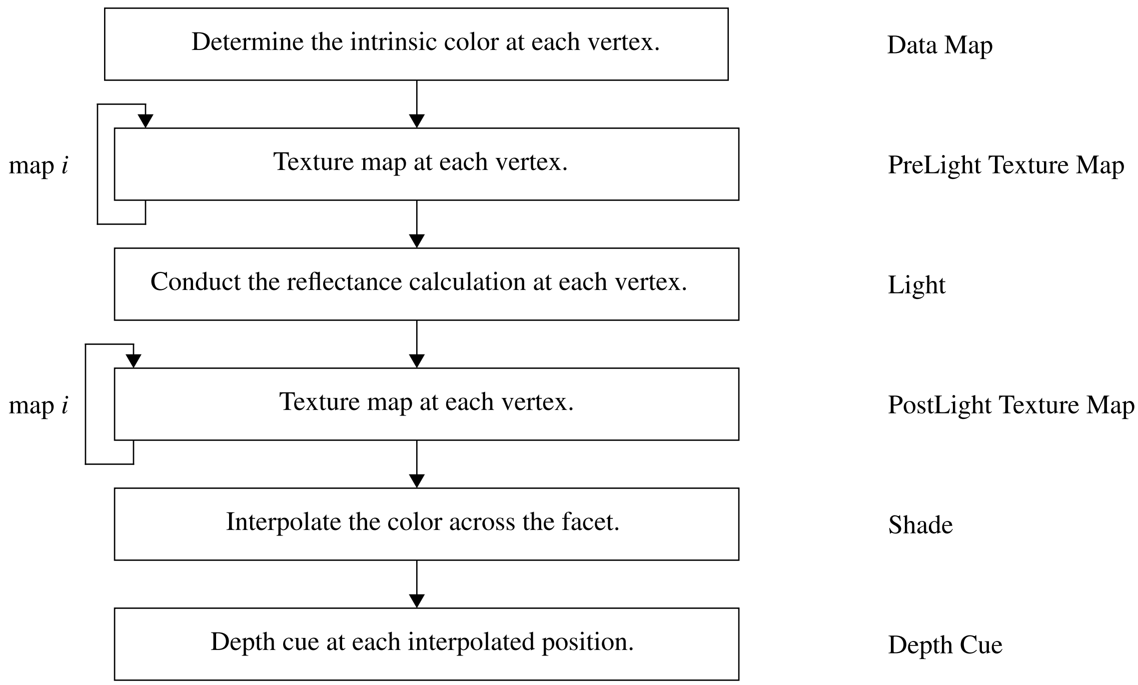


Figure 4.*Surface Interpolation Method Color and Texture Mapping Sample Frequency Pixel*

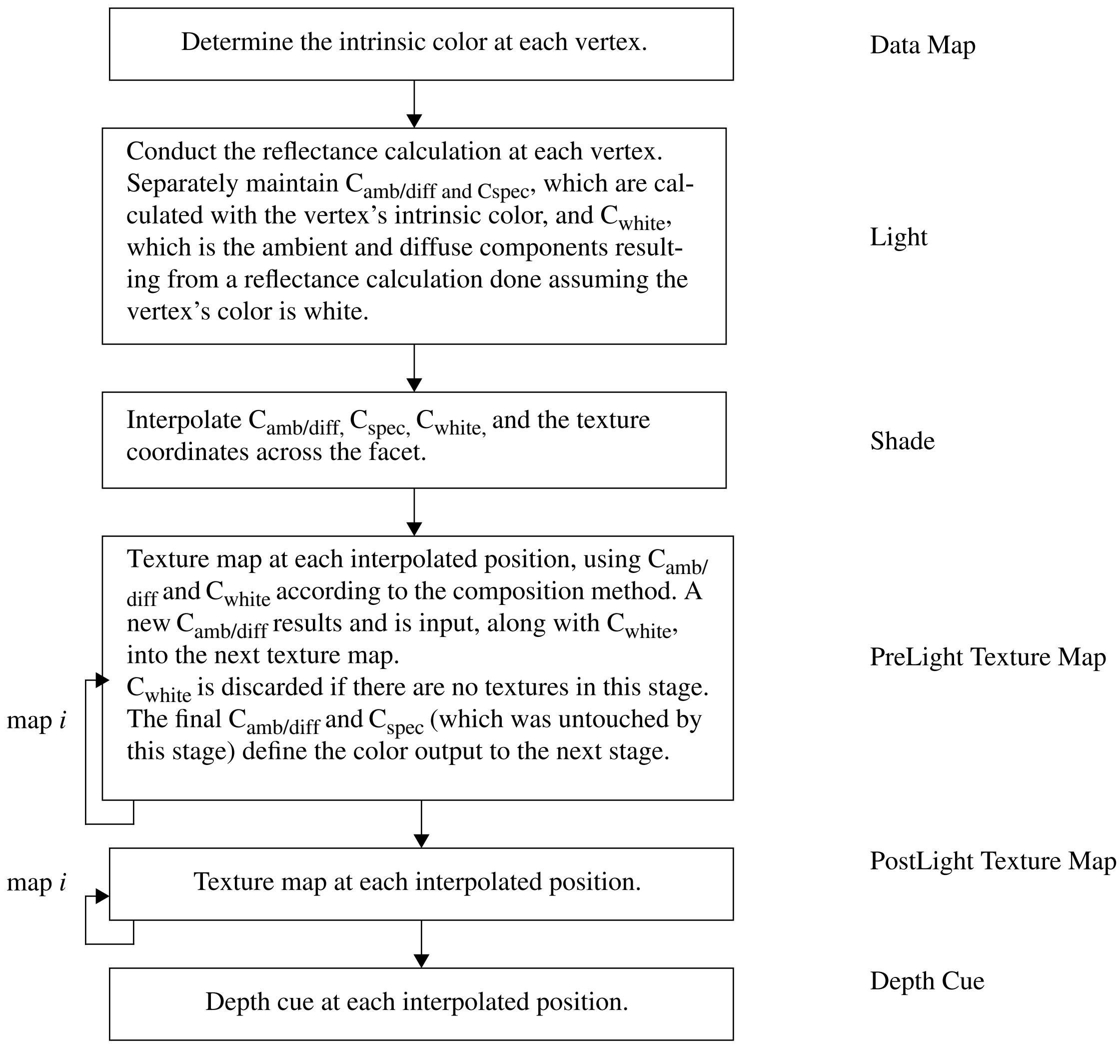


Figure 5.*Surface Interpolation Method DotProduct and Texture Mapping Sample Frequency Pixel*

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

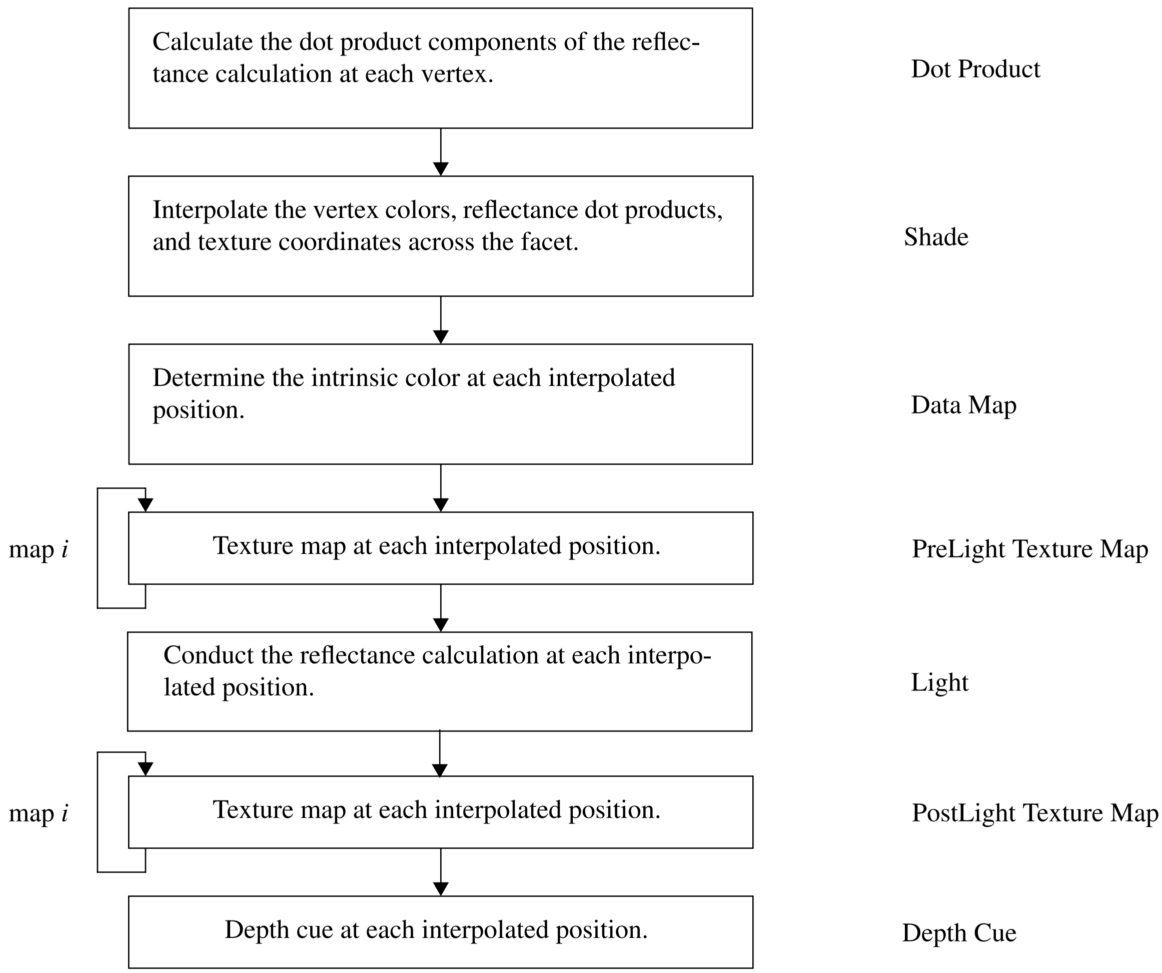
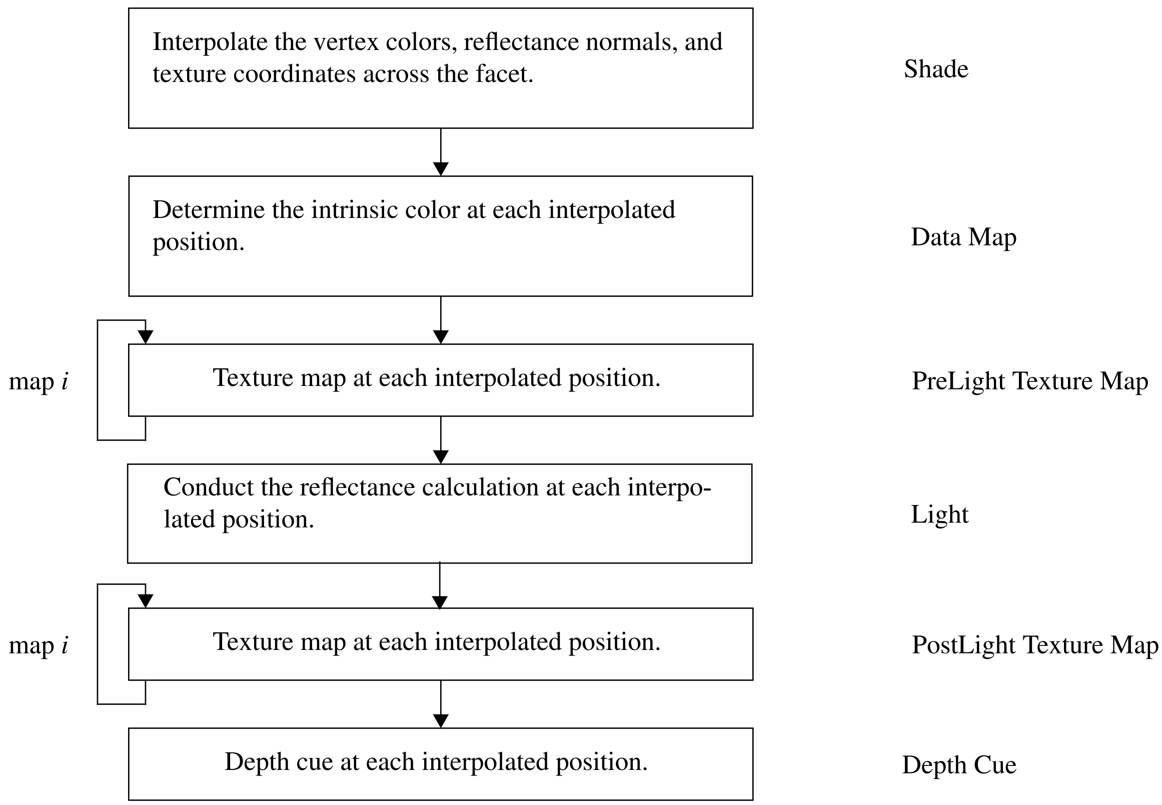


Figure 6.*Surface Interpolation Method Normal and Texture Mapping Sample Frequency Pixel*

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



2.12.7. Antialiasing

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

2.12.8. Color Approximation

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

An entry of the color approximation lookup table (page 442) is always selected by the pipeline state's *color approx index* field, modified by the Color Approximation Index OC (page 304). The primitive's rendering color model color is first transformed to the color model specified by the *color model* field of the color approximation entry. This is the color model then used to convert the color from color model type and components to pixel values. The *approximation method* field specifies how this is done.

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

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

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

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

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

2.12.9. Pixel Color Mode

The *pixel color mode* renderer attribute (page 515) is a switch enabling applications to use pixel values throughout the rendering pipeline as opposed to the color type and components used to define a primitive's intrinsic color and the rendering color model used for intermediate rendering pipeline stages. When the pixel color mode is *On*, PEX assumes the component of all type *Indexed* colors is a pixel value rather than a color lookup table index. PEX converts all colors of other types to pixel values by finding the closest match to the given color in the colormap of the drawable bound to the renderer.

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

2.12.10. Plane Masking

PEX's plane masking is identical to X's plane masking. The pipeline state's *plane mask* field, modified with the Plane Mask OC (page 373), indicates which planes of the image buffer are effected by rendering.

2.12.11. Drawing Function

PEX's drawing functions are identical to X's graphics context functions. The drawing function, which specifies how a pixel value is combined with the pixel value already in the image buffer, is specified by the pipeline state's *drawing function* field, modified with the Drawing Function OC (page 311).

2.12.12. Rendering Pipeline Caveats

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

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

Extensibility

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

It is strongly recommended the following mechanisms be used:

- New requests, and extensions or modifications to existing requests, are implemented through the **PEXEscape** and **PEXEscapeWithReply** requests.
- New primitives, and extensions or modifications to existing primitives, are implemented through the GDP (page 207) and GDP2D OCs (page 207).
- New OCs, and extensions or modifications to existing OCs, other than primitive OCs, are implemented through the GSE OC (page 430).
- New values of an enumerated type class, and new enumerated type classes, are advertised through request **PEXGetEnumeratedTypeInfo** (page 99).
- New implementation-dependent constants are advertised through request **PEXGetImpDepConstants** (page 111).
- Existing events and errors are used since there is no way to programmatically advertise extensions.
- Implementations thoroughly document the interaction of extensions with each other and specified PEX functionality, including additions to the scope of PEX as defined by this chapter.

Compatibility Commitment

It is the client's responsibility to invoke request **PEXGetExtensionInfo** (page 110) to determine the PEX version supported by the server. The client should not attempt to use functionality introduced in a more recent version than the server supports.

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

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