XIElib

Specification

Gary Rogers
AGE Logic, Inc.

This document contains reference pages for each XIElib function.

Public Review Draft
April, 1994

Acknowledgments

XIElib was provided by the X Consortium, with cooperation from AGE Logic, Inc. of San Diego, CA. It is a C subroutine library that provides a low level C binding of all features that are defined by Version 5.0 of the XIE Protocol.

Various parts of the text were borrowed from the Version 5.0 XIE Protocol Reference Manual, edited by Bob Shelley of AGE Logic, Inc.. Bob provided suggestions and comments that were invaluable in preparing this document. Dean Verheiden and Syd Logan, both of AGE Logic, Inc., graciously volunteered to review the original manuscript.

The author would like to thank Ralph Mor of the X Consortium for his careful review of this document. Ralph designed the XIElib and wrote its sample implementation, and his technical advice was indispensable. Adrian Nye, Editor for O'Reilly & Associates, Inc., provided substantive assistance in the preparation of this document, for which the author is very thankful.

Table of Contents

This page describes the format of the reference pages for each XIElib function. Every effort has been made to maintain consistency with the format used in Xlib Reference Manual for Version 11, A. Nye, ed., O'Reilly & Associates, Inc., 1992. The reader is also referred to X Image Extension Protocol Reference Manual, Version 5.0, R. Shelley, ed., 1994, for a complete definition of the XIE protocol.

Name

XieFunctionName - brief description of the function.

Synopsis

The Synopsis section presents the calling syntax for the routine, including the declarations of the arguments and the return type. For example:

```
returntype XieFunctionName (arg1, arg2_ret)
   type1 arg1;
   type2 *arg2_ret;
```

Arguments

The Arguments section describes each of the arguments used by the function. There are two sorts of arguments: arguments to specify data to the function and arguments that return data from the function. An example of each type is shown below:

arg1	Specifies information for `XieFunctionName`. The description for this type of argument always starts with the word "Specifies."
arg2_ret	Returns information from `XieFunctionName`. The description for this type of argument always starts with the word "Returns."

Returns

This section is present when `XieFunctionName` returns a value, and describes what is returned.

Description

The Description section describes what the function does, what it returns, and what events or side-effects it causes. It also may also contain pertinent definitions, algorithms, and tables. A description of each XIE event structure is presented in the section XIElib Events.

Output Attributes

This section, which presents a table of element output attributes, is present if `XieFloFunctionName` specifies an element that produces output data.

Class	Data class of output data - single band (achromatic or index) - triple band (trichromatic)
Type	Data type - constrained (quantization levels is Levels) - unconstrained (quantization levels is unknown)
Width	Width of output (in pixels per band)
Height	Height of output (in pixels per band)
Levels	Depends on type - constrained: number of quantization levels - unconstrained: unknown

Structures

The Structures section contains the C definitions of the XIE-specific data types used by `XieFunctionName` as arguments or return values. It also contains definitions of important constants used by the function.

Errors

The Errors section is present when an action of `XieFunctionName` could generate an error. A table of errors that can be generated and their causes is presented. The full list of errors is presented in the section XIElib Errors.

See Also

This section lists other functions that contain information related to `XieFunctionName`.

Name

XieInitialize - initialize the XIE extension.

Synopsis

```
Status XieInitialize (display, extinfo_ret)
   Display *display;
   XieExtensionInfo **extinfo_ret;
```

Arguments

display	Specifies a connection to an X server.
extinfo_ret	Returns the pointer to an XieExtensionInfo structure, which contains information about the XIE server's capabilities.

Returns

Zero on failure, non-zero on success.

Description

`XieInitialize()` initializes the interface to the XIE extension and returns information about the XIE server's capabilities. `XieInitialize()` should be called to establish version compatibility between client and server prior to making any other XIE request.

If successful, `XieInitialize()` allocates and fills the `XieExtensionInfo` structure as follows:

- The `server_major_rev` and `server_minor_rev` members are set to specify the highest version of the XIE protocol that the server supports. If the server version is higher than the XIElib version, the server will return the lower version, if it supports it.
- The `service_class` member is set to the service-class supported by the XIE server. Service-class defines the recognized image processing service sets supported by the X Image Extension standard; the two service classes currently defined are Full, the entire XIE protocol, and DIS, the Document Image Subset, a proper subset of Full XIE. The `service_class` member can be set to one of the standard values:

  ```
  xieValFull
  xieValDIS
  ```

- The `alignment` member is set to the pixel and scanline alignment for image data supported by the server. Values for this member can be either `xieValAlignable` or `xieValArbitrary`. `xieValAlignable` data units must fit evenly within a byte, or they must fill a byte, or fill a multiple of bytes; `xieValArbitrary` data units may fall at any bit address.
- The `uncnst_mantissa` member is set to the number of bits in the server's floating point format (including the sign bit). If the server uses fixed point, `uncnst_mantissa` is set to 0.
- The `uncnst_min_exp` member is set to the smallest (most negative) value n such that 2^n is representable in the server's unconstrained data format.
- The `uncnst_max_exp` member is set to the largest value n such that $2^n - 1$ is representable in the server's unconstrained data format.
- The `n_cnst_levels` member is the number of items in the list `cnst_levels`.

- The items in the list `cnst_levels` are set to the levels that are “recommended” for constrained data by the server. A value of 0 means 2^{32} levels.
- The `first_event` member is set to the value from which subsequent XIE events values are based.
- The `first_error` member is set to the value from which subsequent XIE error values are based.

The memory allocated to *`extinfo_ret`* is freed when *`display`* is closed via `XCloseDisplay()`; the client should <u>not</u> free this memory.

If not successful, `XieInitialize()` sets ***`extinfo_ret`* to `NULL`.

Structures

```
typedef struct {
   unsigned server_major_rev;
   unsigned server_minor_rev;
   XieServiceClass service_class;
   XieAlignment alignment;
   int uncnst_mantissa;
   int uncnst_min_exp;
   int uncnst_max_exp;
   int n_cnst_levels;
   unsigned long *cnst_levels;
   int major_opcode;
   int first_event;
   int first_error;
} XieExtensionInfo;

/* Definitions of Extension Name and Version Number */
#define xieMajorVersion                5
#define xieMinorVersion                0
#define xieEarliestMinorVersion        0
#define xieLatestMinorVersion          0

/* Definitions of ServiceClass */
#define xieValFull                     1
#define xieValDIS                      2

/* Definitions of Alignment */
#define xieValAlignable                1
#define xieValArbitrary                2
```

Name

XieQueryTechniques - return information about the standard and private techniques that are supported by the server.

Synopsis

```
Status XieQueryTechniques (display, technique_group, ntechniques_ret,
          techniques_ret)
   Display *display;
   XieTechniqueGroup technique_group;
   int *ntechniques_ret;
   XieTechnique **techniques_ret;
```

Arguments

display	Specifies a connection to an X server.
technique_group	Specifies the group of techniques for which the server is to return information.
ntechniques_ret	Returns the number of items in the list of XieTechnique structures.
techniques_ret	Returns the pointer to the list of XieTechnique structures, which contains the information about the selected group of techniques.

Returns

Zero on failure, non-zero on success.

Description

If successful, `XieQueryTechniques()` allocates and fills each `XieTechnique` structure in the list as follows:

- The member `needs_param` is set to `True` if the technique requires additional parameters; `needs_param` is set to `False` if the technique takes no parameters, or it has parameters that are optional. If parameters are optional, they must be totally omitted, or they must all be supplied.
- The member `group` is set to the group the technique belongs to.
- The member `number` is set to the numeric identifier assigned to the technique.
- The member `speed` is set to the server's assessment of the speed of this technique relative to other techniques in the same group, where 0 is slowest and 255 is fastest.
- The member `name` is set to the XIE compliant technique name string.

To free the memory allocated to *techniques_ret*, use `XieFreeTechniques()`.

On failure, *ntechniques_ret* is set to zero and **techniques_ret* is set to `NULL`.

The standard technique group names that can be queried using `XieQueryTechniques()` are shown below:

Technique group	Meaning
`xieValDefault`	select all default techniques
`xieValAll`	select all supported techniques
`xieValColorAlloc`	select color allocation techniques
`xieValConstrain`	select techniques for constraining data
`xieValConvertFromRGB`	select colorspace conversion techniques (for conversion from the RGB colorspace)
`xieValConvertToRGB`	select colorspace conversion techniques (for conversion to the RGB colorspace)
`xieValConvolve`	select techniques for handling convolution edge conditions
`xieValDecode`	select image decoding (decompression) techniques
`xieValDither`	select dithering techniques
`xieValEncode`	select image encoding (compression) techniques
`xieValGamut`	select colorspace conversion gamut compression techniques
`xieValGeometry`	select geometric sampling techniques
`xieValHistogram`	select match-histogram shapes
`xieValWhiteAdjust`	select colorspace conversion white point adjustment techniques

If a vendor defined an additional private technique group, it could be discovered by querying for All groups.

Structures

```
typedef unsigned XieTechniqueGroup;
typedef struct {
   Bool needs_param;
   XieTechniqueGroup group;
   unsigned int number;
   unsigned int speed;
   char *name;
} XieTechnique;

/* Definitions for TechniqueGroups */
#define xieValDefault               0
#define xieValAll                   1
#define xieValColorAlloc            2
#define xieValConstrain             4
#define xieValConvertFromRGB        6
#define xieValConvertToRGB          8
#define xieValConvolve              10
#define xieValDecode                12
#define xieValDither                14
#define xieValEncode                16
#define xieValGamut                 18
#define xieValGeometry              20
#define xieValHistogram             22
#define xieValWhiteAdjust           24
```

Errors

`BadAlloc`	Insufficient resources
`BadValue`	Unknown *`technique_group`*

See Also

XieFreeTechniques().

Name

XieCreateColorList - create a color list.

Synopsis

```
XieColorList XieCreateColorList (display)
   Display *display;
```

Arguments

display	Specifies a connection to an X server.

Returns

The color list identifier.

Description

`XieCreateColorList()` creates a color list resource and returns its color list ID.

The color list created is an unpopulated server resource that can be used to store the list of colors allocated by `XieFloConvertToIndex()`. The Colormap allocations that are recorded in a color list belong to the client that executed the photoflo that populated the resource (this is not necessarily the same client that created the color list). A color list cannot be the target of more than one active photoflo at a time. The contents of a color list may be queried using `XieQueryColorList()`. All allocated cells can be explicitly purged from a color list using `XiePurgeColorList()`. A color list can be destroyed using `XieDestroyColorList()`.

Structures

```
typedef XID XieColorList;
```

Errors

`BadAlloc`	Insufficient resources
`BadIdChoice`	Invalid color list

See Also

XieDestroyColorList(), XiePurgeColorList(), XieQueryColorList(), XieFloConvertToIndex().

Name

XieDestroyColorList - destroy a color list.

Synopsis

```
void XieDestroyColorList (display, color_list)
   Display *display;
   XieColorList color_list;
```

Arguments

display	Specifies a connection to an X server.
color_list	Specifies the color list to be destroyed.

Description

`XieDestroyColorList()` destroys the color list resource identified by *color_list*. Once destroyed, color list ID is no longer valid.

Structures

```
typedef XID XieColorList;
```

Errors

xieErrNoColorlist	Invalid *color_list*

See Also

XieCreateColorList().

Name

XiePurgeColorList - purge all allocated cells from a color list.

Synopsis

```
void XiePurgeColorList (display, color_list)
   Display *display;
   XieColorList color_list;
```

Arguments

display	Specifies a connection to an X server.
color_list	Specifies the color list to be purged.

Description

`XiePurgeColorList()` frees the colors from the specified color list.

Structures

```
typedef XID XieColorList;
```

Errors

BadAccess	Attempt to purge colors when color list is being written by a photoflo
xieErrNoColorlist	Invalid *color_list*

See Also

XieCreateColorList(), XieDestroyColorList(), XieQueryColorList(), XieFloConvertToIndex().

Name

XieQueryColorList - obtain a list of allocated Colormap indices.

Synopsis

```
Status XieQueryColorList (display, color_list, colormap_ret,
            ncolors_ret, colors_ret)
   Display *display;
   XieColorList color_list;
   Colormap *colormap_ret;
   unsigned *ncolors_ret;
   unsigned long **colors_ret;
```

Arguments

display	Specifies a connection to an X server.
color_list	Specifies the color list to query.
colormap_ret	Returns the Colormap from which the colors were allocated.
ncolors_ret	Returns the number of Colormap indices in the list.
colors_ret	Returns the list of allocated Colormap indices.

Returns

Zero on failure, non-zero on success.

Description

`XieQueryColorList()` allocates and returns a list of colors allocated by a ConvertToIndex element.

When there are no colors in color list, a zero status is returned and the value zero is returned for the colormap, and the list of colors is of length zero. The pointer to the list of allocated Colormap indices is set to `NULL`.

To free the memory allocated to *colors_ret*, use `XFree()`.

Structures

```
typedef XID XieColorList;
```

Errors

BadAlloc	Insufficient resources
xieErrNoColorlist	Invalid color list

See Also

XieFloConvertToIndex().

Name

XieCreateLUT - create a lookup table.

Synopsis

```
XieLut XieCreateLUT (display)
   Display *display;
```

Arguments

display	Specifies a connection to an X server.

Returns

The lookup table (LUT) identifier.

Description

`XieCreateLUT()` creates a server resource which is used as a lookup table (LUT) by a Point element. A lookup table consists of one or three single-dimension arrays, each long enough to contain an entry for all possible pixels values in the image data to which the Point element will be applied.

The LUT is populated (or repopulated) with lookup table entries after the successful execution of a photoflo containing an ExportLUT element which targets lut. LUT data can be imported into a photoflo using an ImportLUT element.

Structures

```
typedef XID XieLut;
```

Errors

`BadAlloc`	Insufficient resources
`BadIDChoice`	Invalid LUT

See Also

XieDestroyLUT(), XieFloPoint(), XieFloImportClientLUT(), XieFloImportLUT(), XieFloExportClientLUT(), XieFloExportLUT().

Name

XieDestroyLUT - destroy a lookup table.

Synopsis

```
void XieDestroyLUT (display, lut)
   Display *display;
   XieLut lut;
```

Arguments

display	Specifies a connection to an X server.
lut	Specifies the ID of the LUT to be destroyed.

Description

`XieDestroyLUT()` destroys the lookup table (LUT) identified by *lut*. Once destroyed, LUT ID is no longer valid.

Structures

```
typedef XID XieLut;
```

Errors

`xieErrNoLut`	The value for the *lut* argument does not name a defined LUT

See Also

XieCreateLUT().

Name

XieCreatePhotomap - create a photomap.

Synopsis

```
XiePhotomap XieCreatePhotomap (display)
   Display *display;
```

Arguments

display	Specifies a connection to an X server.

Returns

The photomap identifier.

Description

`XieCreatePhotomap()` creates a photomap, a server resource which stores image data. Photomap data may be rendered for display or used as input to control or modify the rendition of another image.

Photomap attributes are defined when a photoflo containing an ExportPhotomap element populates the photomap with data.

Structures

```
typedef XID XiePhotomap;
```

Errors

`BadAlloc`	Insufficient resources
`BadIdChoice`	Invalid photomap

See Also

XieDestroyPhotomap(), XieQueryPhotomap(), XieFloImportClientPhoto(), XieFloImportPhotomap(), XieFloExportClientPhoto(), XieFloExportPhotomap().

Name

XieDestroyPhotomap - destroy a photomap.

Synopsis

```
void XieDestroyPhotomap (display, photomap)
   Display *display;
   XiePhotomap photomap;
```

Arguments

display	Specifies a connection to an X server.
photomap	Specifies the ID of the photomap to be destroyed.

Description

`XieDestroyPhotomap()` destroys the photomap identified by *photomap*. Once destroyed, the photomap ID is no longer valid. A photomap is the XIE resource used to store image data in the server.

Structures

```
typedef XID XiePhotomap;
```

Errors

`xieErrNoPhotomap`	The value for the *photomap* argument does not name a defined photomap

See Also

XieCreatePhotomap(), XieQueryPhotomap().

Name

XieQueryPhotomap - returns the queriable attributes of a photomap.

Synopsis

```
Status XieQueryPhotomap (display, photomap, populated_ret,
            datatype_ret, class_ret, decode_technique_ret, width_ret,
            height_ret, levels_ret)
   Display *display;
   XiePhotomap photomap;
   Bool *populated_ret;
   XieDataType *datatype_ret;
   XieDataClass *class_ret;
   XieDecodeTechnique *decode_technique_ret;
   XieLTriplet width_ret;
   XieLTriplet height_ret;
   XieLTriplet levels_ret;
```

Arguments

display	Specifies a connection to an X server.
photomap	Specifies the photomap to be queried.
populated_ret	Returns the status of the photomap.
datatype_ret	Returns the type of the data in the photomap.
class_ret	Returns the class of the data in the photomap.
decode_technique_ret	Returns the decode technique required to interpret the data.
width_ret	Returns the width, in pixels per band.
height_ret	Returns the height, in pixels per band.
levels_ret	Returns the number of quantization levels per band.

Returns

Zero on failure, non-zero on success.

Description

A photomap is a server resource which stores image data. `XieQueryPhotomap()` sets *populated_ret* to indicate whether or not *photomap* has been populated with attributes and data. If *populated_ret* is `False`, all remaining fields contain zeros.

datatype_ret shows whether the number of quantization levels is valid (constrained) or unknown (unconstrained), and is set to one of the following standard data type values:

```
xieValConstrained
xieValUnconstrained
```

class_ret is the class of image data (i.e., single-band or triple-band), and is set to one of the following standard data class values:

```
xieValSingleBand
xieValTripleBand
```

`width_ret` and `height_ret` are set to the dimensions of the image data in pixels (per band). `levels_ret` is set to the potential dynamic range, or number of quantization levels (per band). If `datatype_ret` is set to unconstrained, the returned values for levels are zeros. If `class_ret` is `xieValSingleBand`, `width_ret`, `height_ret`, and `levels_ret` for unused bands are returned as zeros.

`decode_technique_ret` is set to the decode technique that will be required to interpret or decompress the data. Decode techniques define the techniques that can be used to interpret uncompressed image data or decode compressed images. `decode_technique_ret` can be set to one of the following standard decode technique values:

```
xieValDecodeUncompressedSingle
xieValDecodeUncompressedTriple
xieValDecodeG31D
xieValDecodeG32D
xieValDecodeG42D
xieValDecodeJPEGBaseline
xieValDecodeJPEGLossless
xieValDecodeTIFF2
xieValDecodeTIFFPackBits
```

If a vendor defined additional private decode techniques, `decode_technique_ret` can be set to the values given to these techniques.

Structures

```
typedef unsigned XieDataClass;
typedef unsigned XieDataType;
typedef unsigned XieDecodeTechnique;
typedef unsigned long XieLTriplet[3];
typedef XID XiePhotomap;

/* Definitions of DataType */
#define xieValConstrained              1
#define xieValUnconstrained            2

/* Definitions of DataClass */
#define xieValSingleBand               1
#define xieValTripleBand               2

/* Definitions for DecodeTechniques */
#define xieValDecodeUncompressedSingle 2
#define xieValDecodeUncompressedTriple 3
#define xieValDecodeG31D               4
#define xieValDecodeG32D               6
#define xieValDecodeG42D               8
#define xieValDecodeJPEGBaseline       10
#define xieValDecodeJPEGLossless       12
#define xieValDecodeTIFF2              14
#define xieValDecodeTIFFPackBits       16
```

Errors

`xieErrNoPhotomap`	The value for the *photomap* argument does not name a defined photomap
`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)

Name

XieCreateROI - Create a Rectangles-Of-Interest.

Synopsis

```
XieRoi XieCreateROI (display)
   Display *display;
```

Arguments

display	Specifies a connection to an X server.

Returns

The ROI (Rectangles-Of-Interest) identifier.

Description

`XieCreateROI()` creates a server ROI (Rectangles-Of-Interest) resource.

Structures

```
typedef XID XieRoi;
```

Errors

`BadAlloc`	Insufficient resources
`BadIDChoice`	Invalid ROI

See Also

XieDestroyROI(), XieFloImportClientROI(), XieFloImportROI(), XieFloExportClientROI(), XieFloExportROI().

Name

XieDestroyROI - destroy a Rectangles-Of-Interest.

Synopsis

```
void XieDestroyROI (display, roi)
   Display *display;
   XieRoi roi;
```

Arguments

display	Specifies a connection to an X server.
roi	Specifies the ID of the ROI to be destroyed.

Description

`XieDestroyROI()` destroys the Rectangles-Of-Interest (ROI) identified by *roi*. Once destroyed, ROI ID is no longer valid.

Structures

```
typedef XID XieRoi;
```

Errors

`xieErrNoROI`	The value for the *roi* argument does not name a defined ROI

See Also

XieCreateROI().

Name

XieCreatePhotospace - create a photospace.

Synopsis

```
XiePhotospace XieCreatePhotospace (display)
   Display *display;
```

Arguments

display	Specifies a connection to an X server.

Returns

The photospace identifier.

Description

`XieCreatePhotospace()` returns a resource-id for a new photospace that can be used to accommodate immediate photoflos instantiated by a client. Any client that needs to instantiate immediate photoflos must create at least one photospace.

Structures

```
typedef XID XiePhotospace;
```

Errors

`BadAlloc`	Insufficient resources
`BadIDChoice`	Invalid photospace

See Also

XieDestroyPhotospace(), XieExecuteImmediate().

Name

XieDestroyPhotospace - destroy a photospace.

Synopsis

```
void XieDestroyPhotospace (display, photospace)
   Display *display;
   XiePhotospace photospace;
```

Arguments

display	Specifies a connection to an X server.
photospace	Specifies the ID of the photospace to be destroyed.

Description

`XieDestroyPhotospace()` destroys a photospace. Prior to destroying the photospace, all photoflos that are currently active in the photospace will be aborted, exported data pending client retrieval will be freed, and the photoflos will be destroyed.

Structures

```
typedef XID XiePhotospace;
```

Errors

`xieErrNoPhotospace`	The value for the *photospace* argument does not name a defined photospace

See Also

XieCreatePhotospace().

Name

XieExecuteImmediate - define and begin execution of an immediate photoflo.

Synopsis

```
void XieExecuteImmediate (display, photospace, flo_id, notify,
            elem_list, elem_count)
   Display *display;
   XiePhotospace photospace;
   unsigned long flo_id;
   Bool notify;
   XiePhotoElement *elem_list;
   int elem_count;
```

Arguments

display	Specifies a connection to an X server.
photospace	Specifies the ID of the photospace to be executed.
flo_id	Specifies a particular instance of the photoflo to be executed.
notify	Specifies whether a `PhotofloDone` event must be sent upon completion.
elem_list	Specifies the import, process, and export elements to be executed.
elem_count	Specifies the number of items in *elem_list*.

Description

`XieExecuteImmediate()` begins the asynchronous execution of an *immediate* photoflo. The server does not save a copy of an immediate photoflo after the photoflo has completed execution and all data exported for the client have been retrieved. An immediate photoflo may therefore not be modified or totally redefined prior to subsequent executions. It is legal to have multiple unique instances of immediate photoflos (and stored photoflos) active concurrently.

The *photospace/flo_id* argument pair specifies the *instance* by which this photoflo will be identified in other requests, events, or errors. *notify* specifies whether a `PhotofloDone` event must be sent upon completion. The `PhotofloDone` event notifies the client that a photoflo has left the active state: it is no longer executing. *elem_list* defines the import, process, and export elements to be executed.

If any clients have blocked themselves during the execution of the photoflo (see `XieAwait()`), they will become unblocked when the photoflo's state changes from active to nonexistent.

Care should be taken that the argument pair *elem_list/elem_count* matches a returned value (an array of `XiePhotoElement` structures) and argument *count* from a call to `XieAllocatePhotofloGraph()`.

Structures

```
typedef XID XiePhotospace;

typedef struct {
   int elemType;
   /* union of ALL element types */
   union {
```

```
        ...
        ...
    } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloElement`	Invalid element type(s) in *`elem_list`*
`xieErrNoFloID`	Invalid *`photospace`*/*`flo_id`* argument pair has been specified
`xieErrNoFlo`	An error has been detected while defining, executing, or accessing a photoflo (See Photoflo Errors).

See Also

XieAwait(), XieAllocatePhotofloGraph().

Name

XieAllocatePhotofloGraph - allocate an array of `XiePhotoElement` structures.

Synopsis

```
XiePhotoElement *XieAllocatePhotofloGraph (count);
   unsigned int count;
```

Arguments

count	Specifies the number of `XiePhotoElement` structures to allocate.

Returns

The array of `XiePhotoElement` structures.

Description

`XieAllocatePhotofloGraph()` allocates and returns a pointer to an array of `XiePhotoElement` structures; each field of each structure in the array is set to zero (0).

To free the memory allocated to the list of `XiePhotoElement` structures, use `XieFreePhotofloGraph()`.

If `XieAllocatePhotofloGraph()` is unable to create an `XiePhotoElement` array , it returns `NULL`.

Structures

```
typedef struct {
   int elemType;
   /* union of ALL element types */
   union {
      ...
      ...
   } data;
} XiePhotoElement;
```

See Also

XieFreePhotofloGraph(), XieCreatePhotoflo(), XieModifyPhotoflo(), XieRedefinePhotoflo(), XieExecutePhotoflo(), XieQueryPhotoflo(), XieDestroyPhotoflo(), XieExecuteImmediate().

Name

XieCreatePhotoflo - create a stored photoflo.

Synopsis

```
XiePhotoflo XieCreatePhotoflo (display, elem_list, elem_count)
   Display *display;
   XiePhotoElement *elem_list;
   int elem_count;
```

Arguments

display	Specifies a connection to an X server.
elem_list	Specifies the defining array of `XiePhotoElement` structures.
elem_count	Specifies the number of `XiePhotoElement` structures in the array.

Returns

The photoflo identifier.

Description

`XieCreatePhotoflo()` creates a *stored* photoflo resource, defines its complete contents using the contents of *elem_list*, sets it in the inactive state, and returns its resource-id. Stored photoflos persist beyond execution and may be modified or totally redefined prior to subsequent executions.

The returned photoflo identifier is a new resource-id which, along with the execution domain used for the photoflo, identifies this photoflo in other requests, events, or errors. *elem_list* defines the import, process, and export elements to be stored for execution. Although resources and parameters are specified at creation, no action is taken to validate them at that time. `XieCreatePhotoflo()` will only store the photoflo's definition: parameter validation is delayed until an execute request is received.

Structures

```
typedef XID XiePhotoflo;

typedef struct {
   int elemType;
   /* union of ALL element types */
   union {
      ...
      ...
   } data;
} XiePhotoElement;
```

Errors

`BadAlloc`	Insufficient resources
`BadIdChoice`	Invalid photoflo
`xieErrNoFloAlloc`	Insufficient resources (e.g. memory) for *elem_list*
`xieErrNoFloElement`	Invalid element type(s) in *elem_list*
`xieErrNoFlo`	An error has been detected while defining, executing, or accessing a photoflo (See Photoflo Errors).

See Also

XieAllocatePhotofloGraph(), XieFreePhotofloGraph(), XieModifyPhotoflo(), XieRedefinePhotoflo(), XieExecutePhotoflo(), XieQueryPhotoflo(), XieDestroyPhotoflo().

Name

XieDestroyPhotoflo - destroy a stored photoflo.

Synopsis

```
void XieDestroyPhotoflo (display, photoflo)
   Display *display;
   XiePhotoflo photoflo;
```

Arguments

display	Specifies a connection to an X server.
photoflo	Specifies the photoflo to be destroyed.

Description

`XieDestroyPhotoflo()` destroys a stored photoflo. If *photoflo* is active, i.e., executing, it is aborted and all exported data that are pending client retrieval are freed prior to destroying *photoflo*.

Structures

```
typedef XID XiePhotoflo;
```

Errors

`xieErrNoPhotoflo`	The value for the *photoflo* argument does not name a defined photoflo

See Also

XieAllocatePhotofloGraph(), XieFreePhotofloGraph(), XieCreatePhotoflo(), XieModifyPhotoflo(), XieRedefinePhotoflo(), XieExecutePhotoflo(), XieQueryPhotoflo().

Name

XieExecutePhotoflo - execute a stored photoflo.

Synopsis

```
void XieExecutePhotoflo (display, photoflo, notify)
   Display *display;
   XiePhotoflo photoflo;
   Bool notify;
```

Arguments

display	Specifies a connection to an X server.
photoflo	Specifies the photoflo to be executed.
notify	Specifies that a `PhotofloDone` event must be sent upon completion.

Description

`XieExecutePhotoflo()` changes a stored photoflo to the active state. Execution is asynchronous. The photoflo returns to the inactive state when execution completes and all data exported for the client have been retrieved. It is legal to have multiple stored photoflos (and immediate photoflos) active concurrently.

notify specifies that a `PhotofloDone` event must be sent upon completion. A `PhotofloDone` event notifies the client that a photoflo has left the active state (it is no longer executing).

Stored photoflos persist beyond execution and may be modified or totally redefined prior to subsequent executions.

Structures

```
typedef XID XiePhotoflo;
```

Errors

`xieErrNoPhotoflo`	The value for the *photoflo* argument does not name a defined photoflo
`xieErrNoFloAccess`	Attempt to execute *photoflo* when it is already active
`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFlo`	An error has been detected while defining, executing, or accessing a photoflo (See Photoflo Errors).

See Also

XieAllocatePhotofloGraph(), XieFreePhotofloGraph(), XieCreatePhotoflo(), XieModifyPhotoflo(), XieRedefinePhotoflo(), XieQueryPhotoflo(), XieDestroyPhotoflo(), XieAbort(), XieAwait(), XieGetClientData(), XiePutClientData().

Name

XieModifyPhotoflo - modify a stored photoflo.

Synopsis

```
void XieModifyPhotoflo (display, photoflo, start, elem_list,
            elem_count)
   Display *display;
   XiePhotoflo photoflo;
   int start;
   XiePhotoElement *elem_list;
   int elem_count;
```

Arguments

display	Specifies a connection to an X server.
photoflo	Specifies the photoflo to be modified.
start	Specifies the index where element replacement is to begin.
elem_list	Specifies an array of elements which will replace existing elements.
elem_count	Specifies the number of items in *elem_list*.

Description

`XieModifyPhotoflo()` allows element parameters of a stored photoflo to be modified. Stored photoflos persist beyond execution and may be modified prior to subsequent executions.

start is the position or index of an element within an array of elements used to specify a photoflo; the first element in the array has a *start* value of one (1).

`XieModifyPhotoflo()` only allows *parameter* modification. No topological changes are allowed: elements cannot be deleted, inserted, or appended.

Structures

```
typedef XID XiePhotoflo;

typedef struct {
   int elemType;
   /* union of ALL element types */
   union {
      ...
      ...
   } data;
} XiePhotoElement;
```

Errors

xieErrNoPhotoflo	The value for the *photoflo* argument does not name a defined photoflo
xieErrNoFloAccess	Attempt to change *photoflo* when it is already active
xieErrNoFloAlloc	Insufficient resources (e.g. memory)
xieErrNoFloElement	Invalid element type(s) in *elem_list* *or* attempt to append additional element(s) to *photoflo*

`xieErrNoFloSource`	An invalid `start` has been specified *or* attempt to change input connections of type XiePhototag in `elem_list`
`xieErrNoFlo`	An error has been detected while defining, executing, or accessing a photoflo (See Photoflo Errors).

See Also

XieAllocatePhotofloGraph(), XieFreePhotofloGraph(), XieCreatePhotoflo(), XieRedefinePhotoflo(), XieExecutePhotoflo(), XieQueryPhotoflo(), XieDestroyPhotoflo().

Name

XieRedefinePhotoflo - redefine a stored photoflo.

Synopsis

```
void XieRedefinePhotoflo (display, photoflo, elem_list, elem_count)
   Display *display;
   XiePhotoflo photoflo;
   XiePhotoElement *elem_list;
   int elem_count;
```

Arguments

display	Specifies a connection to an X server.
photoflo	Specifies the photoflo to be redefined.
elem_list	Specifies an array of elements which will replace all existing elements.
elem_count	Specifies the number of items in *elem_list*.

Description

`XieRedefinePhotoflo()` allows all elements of a stored photoflo to be removed and replaced with a new list. Stored photoflos persist beyond execution and may be totally redefined prior to subsequent executions.

There are no restrictions on changing element types or the array's size.

Structures

```
typedef XID XiePhotoflo;

typedef struct {
   int elemType;
   /* union of ALL element types */
   union {
      ...
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoPhotoflo`	The value for the *photoflo* argument does not name a defined photoflo
`xieErrNoFloAccess`	Attempt to change *photoflo* when it is already active
`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloElement`	Invalid element type(s) in *elem_list*
`xieErrNoFlo`	An error has been detected while defining, executing, or accessing a photoflo (See Photoflo Errors).

See Also

XieAllocatePhotofloGraph(), XieFreePhotofloGraph(), XieCreatePhotoflo(), XieModifyPhotoflo(), XieExecutePhotoflo(), XieQueryPhotoflo(), XieDestroyPhotoflo(), XieExecuteImmediate().

Name

XieQueryPhotoflo - return the current status of a photoflo.

Synopsis

```
XieQueryPhotoflo(display, name_space, flo_id, state_ret,
            data_expected_ret, nexpected_ret, data_available_ret,
            navailable_ret)
   Display *display;
   unsigned long name_space;
   unsigned long flo_id;
   XiePhotofloState *state_ret;
   XiePhototag **data_expected_ret;
   unsigned int *nexpected_ret;
   XiePhototag **data_available_ret;
   unsigned int *navailable_ret;
```

Arguments

display	Specifies a connection to an X server.
name_space	Specifies the execution domain used for the photoflo to query.
flo_id	Specifies a particular instance of the photoflo to query.
state_ret	Returns the state of the photoflo.
data_expected_ret	Returns a list of ImportClient elements.
nexpected_ret	Returns the length of *data_expected_ret*.
data_available_ret	Returns a list of ExportClient elements.
navailable_ret	Returns the length of *data_available_ret*.

Returns

Zero on failure, non-zero on success.

Description

`XieQueryPhotoflo()` will return the current status of a photoflo.

The *name_space/flo_id* argument pair specifies the *instance* that identifies the photoflo which is being queried. *state_ret* indicates the state of the photoflo, and if `XieQueryPhotoflo()` is successful, will return one of the following standard photoflo state values:

```
xieValInactive
xieValActive
xieValNonexistent
```

data_expected_ret is a list of ImportClient elements that are expecting data via `XiePutClientData()`. *data_available_ret* is a list of ExportClient elements from which data are available (via `XieGetClientData()`). Either or both of these lists may be of length zero, indicated by the returned values of *nexpected_ret* and *navailable_ret*.

`XieQueryPhotoflo()` allocates memory for the list of ImportClient elements and the list of ExportClient elements. To free the memory allocated to *data_expected_ret* and *data_available_ret*, use `XFree()`.

Specifying an unknown or invalid *instance* will return a `state_ret` of nonexistent and zero length `data_expected_ret` and `data_available_ret` lists.

Structures

```
typedef unsigned XiePhotofloState;

/* Definitions of PhotofloState */
#define xieValInactive                1
#define xieValActive                  2
#define xieValNonexistent             3
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)

See Also

XieAllocatePhotofloGraph(), XieFreePhotofloGraph(), XieCreatePhotoflo(), XieModifyPhotoflo(), XieRedefinePhotoflo(), XieExecutePhotoflo(), XieQueryPhotoflo(), XieDestroyPhotoflo(), XieGetClientData(), XiePutClientData(), XieExecuteImmediate().

Name

XiePutClientData - send a stream of data to an active photoflo.

Synopsis

```
void XiePutClientData (display, name_space, flo_id, element, final,
            band_number, data, nbytes)
   Display *display;
   unsigned long name_space;
   unsigned long flo_id;
   XiePhototag element;
   Bool final;
   unsigned band_number;
   unsigned char *data;
   unsigned nbytes;
```

Arguments

display	Specifies a connection to an X server.
name_space	Specifies the execution domain used for the photoflo to receive the data.
flo_id	Specifies a particular instance of the photoflo to receive the data.
element	Specifies the element to receive the data.
final	Specifies if the data is the last segment of data to be sent.
band_number	Specifies which band of data is being sent.
data	Specifies a counted list of bytes that comprises the data stream.
nbytes	Specifies the count of bytes that comprises the data stream.

Description

`XiePutClientData()` sends a stream of data to an active photoflo. Since the complete data object may be larger than can fit in a single protocol request, XIE allows the stream to be segmented; the last segment is signaled with a *final* flag.

The organization and contents of the data stream must match the parameters given to the ImportClient element or the results are undefined. An arbitrary amount of image data can be sent per request, whereas for non-image data one or more complete aggregates must be sent per request (e.g. one or more LUT array entries). If too many data are sent (e.g. too many rectangles, or too many scanlines), the unwanted data are discarded. It is an error, however, to send too few data prior to signaling *final*.

For stored photoflos, *name_space* is always ServerIDSpace (the value zero) and *flo_id* is the photoflo's resource-id. For immediate photoflos *name_space* is a photospace resource-id and *flo_id* is a thirty-two (32) bit value that uniquely identifies the instance of the photoflo within *name_space*.

Structures

typedef unsigned XiePhototag;

Errors

`xieErrNoFloAccess`	Executable *photospace*/*flo_id* argument pair not active

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloElement`	Invalid element type specified by *`element`*
`xieErrNoFloID`	Invalid *`photospace/flo_id`* argument pair has been specified
`xieErrNoFloValue`	Invalid *`band_number`* *or* for non-image data, *`data`* contains a partial aggregate

See Also

XieGetClientData(), XieQueryPhotoflo(), XieExecutePhotoflo(), XieFloImportClientPhoto(), XieFloImportClientROI(), XieTecDecodeUncompressedSingle(), XieTecDecodeUncompressedTriple(), XieTecDecodeG31D(), XieTecDecodeG32D(), XieTecDecodeG42D(), XieTecDecodeTIFF2(), XieTecDecodeTIFFPackBits(), XieTecDecodeJPEGBaseline(), XieTecDecodeJPEGLossless().

Name

XieGetClientData - retrieve data from an ExportClient element within an active photoflo.

Synopsis

```
Status XieGetClientData (display, name_space, flo_id, element,
            max_bytes, terminate, band_number, new_state_ret,
            data_ret, nbytes_ret)
   Display *display;
   unsigned long name_space;
   unsigned long flo_id;
   XiePhototag element;
   unsigned max_bytes;
   Bool terminate;
   unsigned band_number;
   XieExportState *new_state_ret;
   unsigned char **data_ret;
   unsigned *nbytes_ret;
```

Arguments

display	Specifies a connection to an X server.
name_space	Specifies the execution domain used for the photoflo from which to retrieve data.
flo_id	Specifies a particular instance of the photoflo from which to retrieve data.
element	Specifies the element from which to retrieve the data.
max_bytes	Specifies the maximum number of bytes that can be sent to the client.
terminate	Specifies whether more data are wanted after this request.
band_number	Specifies which band of data is being retrieved.
new_state_ret	Returns the status of the ExportClient element after this request.
data_ret	Returns a counted list of bytes that comprises the data stream.
nbytes_ret	Returns the count of bytes that comprises the data stream.

Returns

Zero on failure, non-zero on success.

Description

`XieGetClientData()` returns data in a contiguous read-once byte stream, which can be requested in segments that are limited in size by the amount the client desires or the amount of data available from the server. The format of the data depends on the parameters given to the ExportClient element from which the data are requested.

new_state_ret returns the state of the ExportClient element after this request and can be set to one of the following standard export state values:

```
xieValExportDone
xieValExportMore
xieValExportEmpty
xieValExportError
```

If the request is sent to an ExportClient element that either: does not have any data, was terminated by a previous `XieGetClientData()` call, or has already returned all its data (ExportDone sent), the request will return a zero length *data_ret* stream.

Image data are always retrieved from the server as a byte stream, whereas non-image data are always returned by the server as one or more complete aggregates. *max_bytes* is effectively rounded down by the server to the match the nearest aggregate size.

For stored photoflos, *name_space* is always `ServerIDSpace` (the value zero) and *flo_id* is the photoflo's resource-id. For immediate photoflos *name_space* is a photospace resource-id and *flo_id* is a thirty-two (32) bit value that uniquely identifies the instance of the photoflo within *name_space*.

To free the memory allocated to *data_ret*, use `XFree()`.

Structures

typedef unsigned XieExportState;
typedef unsigned XiePhototag;

```
/* Definitions of ExportState */
#define xieValExportDone             1
#define xieValExportMore             2
#define xieValExportEmpty            3
#define xieValExportError            4
```

Errors

`xieErrNoFloAccess`	Executable *photospace*/*flo_id* argument pair not active
`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloElement`	Invalid element type specified by *element*
`xieErrNoFloID`	Invalid *photospace*/*flo_id* argument pair has been specified
`xieErrNoFloValue`	Invalid *band_number*

See Also

XiePutClientData(), XieQueryPhotoflo(), XieExecutePhotoflo(), XieFloExportClientHistogram(), XieFloExportClientLUT(), XieFloExportClientPhoto(), XieFloExportClientROI(), XieTecEncodeUncompressedSingle(), XieTecEncodeUncompressedTriple(), XieTecEncodeG31D(), XieTecEncodeG32D(), XieTecEncodeG42D(), XieTecEncodeJPEGBaseline(), XieTecEncodeJPEGLossless(), XieTecEncodeTIFF2(), XieTecEncodeTIFFPackbits().

Name

XieAbort - prematurely terminate execution of a photoflo.

Synopsis

```
void XieAbort (display, name_space, flo_id);
   Display *display;
   unsigned long name_space;
   unsigned long flo_id;
```

Arguments

display	Specifies a connection to an X server.
name_space	Specifies the execution domain used for the photoflo to abort.
flo_id	Specifies a particular instance of the photoflo to abort.

Description

`XieAbort()` will prematurely terminate execution of the photoflo specified by *name_space* and *flo_id*. Any output from the photoflo that is pending client retrieval is freed. Stored photoflos are returned to the inactive state; immediate photoflos are destroyed.

If the photoflo specified by *name_space* and *flo_id* is either invalid or not active no action is taken; it is not an error, and nothing is destroyed.

For stored photoflos, *name_space* is always `ServerIDSpace` (the value zero) and *flo_id* is the photoflo's resource-id. For immediate photoflos *name_space* is a photospace resource-id and *flo_id* is a thirty-two (32) bit value that uniquely identifies the instance of the photoflo within *name_space*.

See Also

XieExecutePhotoflo().

Name

XieAwait - block all further requests for this client connection from being honored by the server while the photoflo is active.

Synopsis

```
void XieAwait (display, name_space, flo_id);
   Display *display;
   unsigned long name_space;
   unsigned long flo_id;
```

Arguments

display	Specifies a connection to an X server.
name_space	Specifies the execution domain used for the photoflo to block requests.
flo_id	Specifies a particular instance of the photoflo to block requests.

Description

`XieAwait()` blocks all further requests for this client connection from being honored by the server while the photoflo, specified by *name_space* and *flo_id*, is active. When the photoflo transitions from the active state, blocked requests are allowed to be processed in the order received.

If the photoflo specified by *name_space* and *flo_id* is either invalid or not active no action is taken; it is not an error, and the client is not blocked.

For stored photoflos, *name_space* is always ServerIDSpace (the value zero) and *flo_id* is the photoflo's resource-id. For immediate photoflos *name_space* is a photospace resource-id and *flo_id* is a thirty-two (32) bit value that uniquely identifies the instance of the photoflo within *name_space*.

Warning

Calling `XieAwait()` before sending all import data or before retrieving all export data, will block the client from sending or retrieving the remaining data. This will also prevent completion of the photoflo and prevent any and all protocol requests from this client from being honored. This deadlock can only be broken by another client completing or aborting the photoflo (to release the Await), or by breaking the client connection.

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)

See Also

XieExecuteImmediate().

Name

XieFloImportClientLUT - specify an ImportClientLUT element and set its parameters.

Synopsis

```
void XieFloImportClientLUT (element, data_class, band_order, length,
            levels)
   XiePhotoElement *element;
   XieDataClass data_class;
   XieOrientation band_order;
   XieLTriplet length;
   XieLevels levels;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
data_class	Specifies the number of lookup arrays to expect.
band_order	Specifies the order of triple band arrays.
length	Specifies the number of entries per array.
levels	Specifies the number of quantization levels represented per array.

Description

An ImportClientLUT element accepts lookup table data from the protocol stream. The transport of data through the protocol stream is accomplished using `XiePutClientData()`. This data is accepted by the Point, ExportLUT, and ExportClientLUT elements.

data_class, which specifies the number of lookup arrays to expect, can be set to one of the following standard data class values:

```
xieValSingleBand
xieValTripleBand
```

The *length* of each array should match the number of source image levels that will be remapped through the array. When a triple band image is to be remapped through a single band array, the *length* of the array should match the product of the source image levels of all three bands; in this case, *band_order* specifies the order in which pixels from a triple band image should be combined to form indices for a single band array. *band_order* can be set to one of the following standard orientation values:

```
xieValLSFirst
xieValMSFirst
```

The least significant band of trichromatic data is the first band mentioned in the common name of the colorspace: red is the least significant band of RGB data. When one LUT array is used with triple band data, the algorithm for computing combined array indices, based on *band_order*, is:

LUT band-order	LUT indexing algorithm for combining pixel values
LSFirst	index = value[0] + value[1] x levels[0] + value[2] x levels[0] x levels[1]
MSFirst	index = value[2] + value[1] x levels[2] + value[0] x levels[2] x levels[1]

When three LUT arrays are used, *band_order* specifies whether this band corresponds with the least significant or most significant LUT array. Each array is transported as a separate data stream:

band	LSFirst	MSFirst
0	Red array	Blue array
1	Green array	Green array
2	Blue array	Red array

Structures

`XieFloImportClientLUT()` sets the `XiePhotoElement` structure field `elemType` to `xieElemImportClientLUT`, which identifies the element as an ImportClientLUT, and sets the fields of the member structure `ImportClientLUT` using the arguments in the argument list.

```
typedef unsigned XieDataClass;
typedef unsigned XieOrientation;
typedef unsigned long XieLTriplet[3];
typedef unsigned long XieLevels[3];

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XieDataClass data_class;
         XieOrientation band_order;
         XieLTriplet length;
         XieLevels levels;
      } ImportClientLUT;
      ...
   } data;
} XiePhotoElement;

/* Definitions of DataClass */
#define xieValSingleBand              1
#define xieValTripleBand              2

/* Definitions of Orientation Types */
#define xieValLSFirst                 1
#define xieValMSFirst                 2
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	*levels* is incompatible with the server's depth handling capabilities
`xieErrNoFloValue`	Invalid *data_class* or *data_class*

See Also

XiePutClientData(), XieQueryPhotoflo(), XieFloExportLUT(), XieFloExportClientLUT(), XieFloPoint(), XieFloImportLUT().

Name

XieFloImportClientPhoto - specify an ImportClientPhoto element and set its parameters.

Synopsis

```
void XieFloImportClientPhoto (element, data_class, width, height,
            levels, notify, decode_tech, decode_param)
   XiePhotoElement *element;
   XieDataClass data_class;
   XieLTriplet width;
   XieLTriplet height;
   XieLevels levels;
   Bool notify;
   XieDecodeTechnique decode_tech;
   XiePointer decode_param;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
data_class	Specifies whether the data is single band or triple band.
width	Specifies the width of the image in pixels per band.
height	Specifies the height of the image in pixels per band.
levels	Specifies the number of quantization levels per band.
notify	Specifies whether to enable sending `DecodeNotify` events.
decode_tech	Specifies the decode technique required to interpret the image.
decode_param	Specifies the list of additional parameters required by *decode_tech*.

Description

An ImportClientPhoto element accepts image data from the protocol stream. This data may be processed for display or used as *process domain* data. A *process domain* is inserted in many element definitions and is used to restrict the element's processing to a subset of the source data pixels. The attributes and organization of the expected data stream are fully specified by the parameters. The actual transport of image data through the protocol stream is requested using XiePutClientData().

notify enables `DecodeNotify` events to be sent if anomalies are encountered while interpreting the imported image data: either an error has been encountered while decoding an image, or the image data received does not satisfy the expected dimensions.

Only constrained data can be sent through the protocol stream; therefore, *levels* must be valid.

data_class specifies whether the data is single band or triple band and can be set to one of the following standard data class values:

```
xieValSingleBand
xieValTripleBand
```

Decode techniques define the techniques that can be used to interpret uncompressed image data or decode compressed images. *decode_tech* can be assigned one of the following standard decode technique values:

```
xieValDecodeUncompressedSingle
xieValDecodeUncompressedTriple
xieValDecodeG31D
xieValDecodeG32D
xieValDecodeG42D
xieValDecodeJPEGBaseline
xieValDecodeJPEGLossless
xieValDecodeTIFF2
xieValDecodeTIFFPackBits
```

If a vendor defined additional private decode techniques, the values given to these techniques can be assigned to *decode_tech*.

Output Attributes

Class	class of imported image
Type	constrained
Width	width of imported image (in pixels)
Height	height of imported image (in pixels)
Levels	levels of imported image

Structures

`XieFloImportClientPhoto()` sets the `XiePhotoElement` structure field `elemType` to `xieElemImportClientPhoto`, which identifies the element as an ImportClientPhoto, and sets the fields of the member structure `ImportClientPhoto` using the arguments in the argument list.

```
typedef unsigned XieDataClass;
typedef unsigned XieDecodeTechnique;
typedef unsigned long XieLTriplet[3];
typedef unsigned long XieLevels[3];

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XieDataClass data_class;
         XieLTriplet width;
         XieLTriplet height;
         XieLevels levels;
         Bool notify;
         XieDecodeTechnique decode_tech;
         XiePointer decode_param;
      } ImportClientPhoto;
      ...
   } data;
} XiePhotoElement;

/* Definitions of DataClass */
#define xieValSingleBand               1
#define xieValTripleBand               2

/* Definitions for DecodeTechniques */
#define xieValDecodeUncompressedSingle 2
#define xieValDecodeUncompressedTriple 3
#define xieValDecodeG31D               4
#define xieValDecodeG32D               6
#define xieValDecodeG42D               8
```

```
#define xieValDecodeJPEGBaseline       10
#define xieValDecodeJPEGLossless       12
#define xieValDecodeTIFF2              14
#define xieValDecodeTIFFPackBits       16
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	*`levels`* is incompatible with the server's depth handling capabilities
`xieErrNoFloTechnique`	Invalid *`decode_tech`* or *`decode_param`*
`xieErrNoFloValue`	Invalid *`width`*, *`height`*, *`levels`* (zero) *or* invalid *`data_class`*

See Also

XiePutClientData(), XieQueryPhotoflo().
XieTecDecodeUncompressedSingle(), XieTecDecodeUncompressedTriple(), XieTecDecodeG31D(), XieTecDecodeG32D(), XieTecDecodeG42D(), XieTecDecodeTIFF2(), XieTecDecodeTIFFPackBits(), XieTecDecodeJPEGBaseline(), XieTecDecodeJPEGLossless().
XieFloExportPhotomap(), XieFloExportClientPhoto(), XieFloImportPhotomap().

Name

XieFloImportClientROI - specify an ImportClientROI element and set its parameters.

Synopsis

```
void XieFloImportClientROI (element, rectangles);
   XiePhotoElement *element;
   unsigned int rectangles;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
rectangles	Specifies the number of rectangles expected.

Description

An ImportClientROI element accepts a list of rectangles from the protocol stream. These data can be used as input to a *process domain* or an ExportROI or ExportClientROI element. A *process domain* is inserted in many element definitions and is used to restrict the element's processing to a subset of the source data pixels. The actual transport of data through the protocol stream is accomplished using `XiePutClientData()` (the *band_number* parameter of `XiePutClientData()` is ignored).

Structures

`XieFloImportClientROI()` sets the `XiePhotoElement` structure field `elemType` to `xieElemImportClientROI`, which identifies the element as an ImportClientROI, and sets the fields of the member structure `ImportClientROI` using the arguments in the argument list.

```
typedef struct {
   int elemType;
   union {
      ...
      struct {
         unsigned int rectangles;
      } ImportClientROI;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)

See Also

XiePutClientData(), XieQueryPhotoflo().
XieFloExportROI(), XieFloExportClientROI(), XieFloImportROI().

Name

XieFloImportDrawable - specify an ImportDrawable element and set its parameters.

Synopsis

```
void XieFloImportDrawable (element, drawable, src_x, src_y, width,
            height, fill, notify)
   XiePhotoElement *element;
   Drawable drawable;
   int src_x;
   int src_y;
   unsigned int width;
   unsigned int height;
   unsigned long fill;
   Bool notify;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
drawable	Specifies the `Drawable` resource supplying the data.
src_x	Specifies the left corner of the region of the data to be imported.
src_y	Specifies the upper corner of the region of the data to be imported.
width	Specifies the width of the region of the data to be imported.
height	Specifies the height of the region of the data to be imported.
fill	Specifies the Colormap index to use for all regions that are obscured.
notify	Specifies whether to enable sending `ImportObscured` events.

Description

An ImportDrawable element allows access to data existing in a `Drawable`. This data may be processed for display or, if *`drawable`* is one bit deep, used as *process domain* data. A *process domain* is inserted in many element definitions and is used to restrict the element's processing to a subset of the source data pixels.

`notify` enables `ImportObscured` events to be sent if data for one or more regions of a Window are obscured and cannot be retrieved from backing store. The arguments *`src_x`*, *`src_y`*, *`width`*, and *`height`* specify the region of data to be imported from *`drawable`*, where *`src_x`* and *`src_y`* define the upper-left corner of the region.

Output Attributes

Class	single band
Type	constrained
Width	*`width`*
Height	*`height`*
Levels	2^{depth} (i.e. drawable depth)

Structures

`XieFloImportDrawable()` sets the `XiePhotoElement` structure field `elemType` to `xieElemImportDrawable`, which identifies the element as an ImportDrawable, and sets the fields of the member structure `ImportDrawable` using the arguments in the argument list.

```
typedef struct {
   int elemType;
   union {
      ...
      struct {
         Drawable drawable;
         int src_x;
         int src_y;
         unsigned int width;
         unsigned int height;
         unsigned long fill;
         Bool notify;
      } ImportDrawable;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDrawable`	Invalid *drawable*
`xieErrNoFloValue`	Invalid region *width*, *height*, *src_x*, *src_y*

Name

XieFloImportDrawablePlane - specify an ImportDrawablePlane element and set its parameters.

Synopsis

```
void XieFloImportDrawablePlane (element, drawable, src_x, src_y,
            width, height, fill, bit_plane, notify)
   XiePhotoElement *element;
   Drawable drawable;
   int src_x;
   int src_y;
   unsigned int width;
   unsigned int width;
   unsigned long fill;
   unsigned long bit_plane;
   Bool notify;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
drawable	Specifies the `Drawable` resource supplying the data.
src_x	Specifies the left corner of the region of the data to be imported.
src_y	Specifies the upper corner of the region of the data to be imported.
width	Specifies the width of the region of the data to be imported.
height	Specifies the height of the region of the data to be imported.
fill	Specifies the Colormap index to use for all regions that are obscured.
bit_plane	Specifies the plane to be imported from *drawable*.
notify	Specifies whether to enable sending `ImportObscured` events.

Description

An ImportDrawablePlane event allows access to a single plane of data existing in a `Drawable`. This data may be processed for display or used as *process domain* data. A *process domain* is inserted in many element definitions and is used to restrict the element's processing to a subset of the source data pixels.

notify enables `ImportObscured` events to be sent if data for one or more regions of a Window are obscured and cannot be retrieved from backing store. The arguments *src_x*, *src_y*, *width*, and *height* specify the region of data to be imported from *drawable*, where *src_x* and *src_y* define the upper-left corner of the region.

bit_plane must have exactly one bit set to one (1), and the value of *bit_plane* must be less than 2^n, where n is the depth of *drawable*. This single bit selects the corresponding bit to be extracted from pixels within *drawable*.

Output Attributes

Class	single band
Type	constrained
Width	*width*
Height	*height*
Levels	2

Structures

`XieFloImportDrawablePlane()` sets the `XiePhotoElement` structure field `elemType` to `xieElemImportDrawablePlane`, which identifies the element as an ImportDrawablePlane, and sets the fields of the member structure `ImportDrawablePlane` using the arguments in the argument list.

```
typedef struct {
   int elemType;
   union {
      ...
      struct {
         Drawable drawable;
         int src_x;
         int src_y;
         unsigned int width;
         unsigned int height;
         unsigned long fill;
         unsigned long bit_plane;
         Bool notify;
      } ImportDrawablePlane;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDrawable`	Invalid *drawable*
`xieErrNoFloValue`	Invalid *bit_plane* or region *width*, *height*, *src_x*, *src_y*

Name

XieFloImportLUT - specify an ImportLUT element and set its parameters.

Synopsis

```
void XieFloImportLUT (element, lut)
   XiePhotoElement *element;
   XieLut lut;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
lut	Specifies the LUT resource supplying the lookup table.

Description

An ImportLUT element allows access to lookup table data existing in a LUT resource. These data are accepted by the Point, ExportLUT, and ExportLUT elements.

Attributes of the lookup table data are inherited from `lut`.

Structures

`XieFloImportLUT()` sets the `XiePhotoElement` structure field `elemType` to `xieElemImportLUT`, which identifies the element as an ImportLUT, and sets the fields of the member structure `ImportLUT` using the arguments in the argument list.

```
typedef XID XieLut;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XieLut lut;
      } ImportLUT;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAccess`	Attempt to import from `lut` before it has been populated
`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloLUT`	An unknown `lut` has been specified

Name

XieFloImportPhotomap - specify an ImportPhotomap element and set its parameters.

Synopsis

```
void XieFloImportPhotomap (element, photomap, notify)
   XiePhotoElement *element;
   XiePhotomap photomap;
   Bool notify;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
photomap	Specifies the photomap resource supplying image data.
notify	Specifies whether to enable sending `DecodeNotify` events.

Description

An ImportPhotomap element allows access to image data existing in a photomap; a photomap is a server resource which can be used to store image data. This data may be processed for display or used as process domain data. A *process domain* is inserted in many element definitions and is used to restrict the element's processing to a subset of the source data pixels.

notify enables `DecodeNotify` events to be sent if anomalies are encountered while decoding compressed data: either an error has been encountered while decoding an image, or the image data received does not satisfy the expected dimensions.

Attributes of the source data are inherited from *photomap*.

Output Attributes

Class	same as *photomap*
Type	same as *photomap*
Width	same as *photomap*
Height	same as *photomap*
Levels	same as *photomap*

Structures

`XieFloImportPhotomap()` sets the `XiePhotoElement` structure field `elemType` to `xieElemImportPhotomap`, which identifies the element as an ImportPhotomap, and sets the fields of the member structure `ImportPhotomap` using the arguments in the argument list.

```
typedef XID XiePhotomap;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhotomap photomap;
         Bool notify;
      } ImportPhotomap;
      ...
```

```
    } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAccess`	Attempt to import from *photomap* before it has been populated
`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloPhotomap`	An unknown *photomap* has been specified

Name

XieFloImportROI - specify an ImportROI element and set its parameters.

Synopsis

```
void XieFloImportROI (element, roi)
   XiePhotoElement *element;
   XieRoi roi;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
roi	Specifies the ID of the ROI supplying the list-of-rectangles.

Description

An ImportROI element allows access to a list-of-rectangles existing in a ROI resource. This data may be referenced by a *process domain*. A *process domain* is inserted in many element definitions and is used to restrict the element's processing to a subset of the source data pixels.

Structures

`XieFloImportROI()` sets the `XiePhotoElement` structure field `elemType` to `xieElemImportROI`, which identifies the element as an ImportROI, and sets the fields of the member structure `ImportROI` using the arguments in the argument list.

```
typedef XID XieRoi;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XieRoi roi;
      } ImportROI;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAccess`	Attempt to import from *roi* before it has been populated
`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloPhotomap`	An unknown *roi* has been specified

Name

XieFloArithmetic - specify an Arithmetic element and set its parameters.

Synopsis

```
void XieFloArithmetic (element, src1, src2, domain, constant,
            operator, band_mask)
   XiePhotoElement *element;
   XiePhototag src1;
   XiePhototag src2;
   XieProcessDomain *domain;
   XieConstant constant;
   XieArithmeticOp operator;
   unsigned int band_mask;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src1	Specifies the first data source.
src2	Specifies the second data source.
domain	Specifies the subset of source data that will be operated on.
constant	Specifies the constant data source.
operator	Specifies the arithmetic operation to be performed.
band_mask	Specifies which bands are to be operated upon.

Description

An Arithmetic element produces output data by performing an addition, subtraction, minimum, or maximum operation between two data sources or between a single data source and a constant. Furthermore, multiplication, division, or gamma correction may by applied to a single data source.

When two sources are involved, *src1* and *src2* are the *phototags* of the elements supplying source data (*constant* is ignored). A phototag is the position or index of an element within an array of elements used to specify a photoflo; the first element in the array has a phototag value of one (1). If the operation is to involve a constant, *src1* is one operand, *src2* must be zero, and *constant* is used as the other operand.

When two sources are involved, all attributes, other than *width* and *height*, must match; all output attributes are inherited from *src*1.

In order to specify a subset of source data that will be operated on, the `phototag`, `offset_x`, and `offset_y` fields of the `XieProcessDomain` structure pointed to by *domain* must be supplied; XIElib does not provide a convenience function to create and/or fill in an `XieProcessDomain` structure. If the entire source data is to be operated on, a pointer to an `XieProcessDomain` structure must still be provided, with the `phototag` field set to zero (0); the `offset_x` and `offset_y` fields are ignored.

Only bands selected by *band_mask* are subject to processing. Other bands present in the image are passed through to the output. For example, a *band_mask* of 001_2 indicates that only the “least significant band” would be processed; operating on all bands requires a *band_mask* of 111_2. Using *band_mask* to select source data that have two (2) or less *levels* is not permitted.

Pixel computations that would lead to errors, will yield valid server-dependent values (e.g. dividing by a constrained pixel value of zero might result in a value of *levels*-1).

The valid operations for the Arithmetic process element are:

Operator	**src1 (operator) src2**	**src1 (operator) constant**
`xieValAdd`	src1 + src2	src1 + constant
`xieValSub`	src1 - src2	src1 - constant
`xieValSubRev`	src2 - src1	constant - src1
`xieValMul`	src1 * constant	
`xieValDiv`	src1 / constant	
`xieValDivRev`	constant / src1	
`xieValMin`	minimum(src1, src2)	minimum(src1, constant)
`xieValMax`	maximum(src1, src2)	maximum(src1, constant)
`xieValGamma` (constrained)	$(levels - 1) * ((src1 / (levels - 1))^{constant})$	
`xieValGamma` (unconstrained)	$src1^{constant}$	

Output Attributes

Class	same as *src1*
Type	same as *src1*
Width	same as *src1*
Height	same as *src1*
Levels	same as *src1*

Structures

`XieFloArithmetic()` sets the `XiePhotoElement` structure field `elemType` to `xieElemArithmetic`, which identifies the element as an Arithmetic, and sets the fields of the member structure `Arithmetic` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
typedef float XieConstant[3];
typedef unsigned XieArithmeticOp;
typedef struct {
    int offset_x;
    int offset_y;
    XiePhototag phototag;
} XieProcessDomain;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src1;
         XiePhototag src2;
         XieProcessDomain domain;
         XieConstant constant;
         XieArithmeticOp operator;
         unsigned int band_mask;
```

```
        } Arithmetic;
        ...
    } data;
} XiePhotoElement;

/* Definitions of ArithmeticOperations */
#define xieValAdd                       1
#define xieValSub                       2
#define xieValSubRev                    3
#define xieValMul                       4
#define xieValDiv                       5
#define xieValDivRev                    6
#define xieValMin                       7
#define xieValMax                       8
#define xieValGamma                     9
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
xieErrNoFloDomain	Invalid *domain*
`xieErrNoFloMatch`	Class, type or levels differ between *src1* and *src2 or* selected data source are bitonal
`xieErrNoFloOperator`	Invalid *operator*
`xieErrNoFloSource`	Invalid *src1* or *src2 or* *src2* has been specified with a monadic operator

Name

XieFloBandCombine - specify a BandCombine element and set its parameters.

Synopsis

```
void XieFloBandCombine (element, src1, src2, src3)
   XiePhotoElement *element;
   XiePhototag src1;
   XiePhototag src2;
   XiePhototag src3;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src1	Specifies the first element supplying source data.
src2	Specifies the second element supplying source data.
src3	Specifies the third element supplying source data.

Description

A BandCombine element merges three single band data sources to produce a triple band result. The arguments *src1, src2,* and *src3* must be of the same type, and each source must be single band. Other attributes which are taken from the individual sources may differ. The output will be triple band.

Output Attributes

Class	triple band
Type	same as *src1*
Width	same as *src*s
Height	same as *src*s
Levels	same as *src*s

Structures

`XieFloBandCombine()` sets the `XiePhotoElement` structure field `elemType` to `xieElemBandCombine`, which identifies the element as a BandCombine, and sets the fields of the member structure `BandCombine` using the arguments in the argument list.

```
typedef unsigned XiePhototag;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src1;
         XiePhototag src2;
         XiePhototag src3;
      } BandCombine;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	A source has more than one band *or* type differs between sources
`xieErrNoFloSource`	Invalid *`src1`*, *`src2`*, or *`src3`*

Name

XieFloBandExtract - specify a BandExtract element and set its parameters.

Synopsis

```
void XieFloBandExtract (element, src, levels, bias, coefficients)
   XiePhotoElement *element;
   XiePhototag src;
   unsigned int levels;
   double bias;
   XieConstant coefficients;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
levels	Specifies the number of quantization levels for the output.
bias	Specifies the value to be added to each output pixel.
coefficients	Specifies the proportion of each source band pixel to be used.

Description

A BandExtract element produces single band output data from a triple band source by multiplying a pixel value from each source band by its corresponding *coefficient* and then summing the results with the *bias* value.

coefficients is a three (3) element array that determines the proportion of each source band pixel that is used to form the output. *levels* is used as the levels attribute of the output data if the *src* data are constrained, otherwise it is ignored.

The source data must be triple band, and all bands must have equal dimensions. The output data will be single band, with levels taken from the *levels* parameter, if the data type is constrained. All other attributes are inherited from *src*.

Output Attributes

Class	single band
Type	same as *src*
Width	same as *src*
Height	same as *src*
Levels	*levels* or unknown

Structures

`XieFloBandExtract()` sets the `XiePhotoElement` structure field `elemType` to `xieElemBandExtract`, which identifies the element as a BandExtract, and sets the fields of the member structure `BandExtract` using the arguments in the argument list.

```
typedef float XieConstant[3];
typedef unsigned XiePhototag;

typedef struct {
   int elemType;
   union {
```

```
        ...
        struct {
           XiePhototag src;
           unsigned int levels;
           float bias;
           XieConstant coefficients;
        } BandExtract;
        ...
    } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	*`src`* is not triple band *or* unequal inter-band dimensions
`xieErrNoFloSource`	Invalid *`src`*

Name

XieFloBandSelect - specify a BandSelect element and set its parameters.

Synopsis

```
void XieFloBandSelect (element, src, band_number)
   XiePhotoElement *element;
   XiePhototag src;
   unsigned int band_number;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
band_number	Specifies which *src* band is to be selected to provide the output data.

Description

A BandSelect element produces single band output data by selecting a single band from a triple band source.

Output Attributes

Class	single band
Type	same as *src*
Width	same as band selected from *src*
Height	same as band selected from *src*
Levels	same as band selected from *src*

Structures

`XieFloBandSelect()` sets the `XiePhotoElement` structure field `elemType` to `xieElemBandSelect`, which identifies the element as a BandSelect, and sets the fields of the member structure `BandSelect` using the arguments in the argument list.

```
typedef unsigned XiePhototag;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         unsigned int band_number;
      } BandSelect;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	*src* is not triple band
`xieErrNoFloSource`	Invalid *src*
`xieErrNoFloValue`	Invalid *band_number*

Name

XieFloBlend - specify a Blend element and set its parameters.

Synopsis

```
void XieFloBlend (element, src1, src2, src_constant, alpha,
            alpha_const, domain, band_mask)
   XiePhotoElement *element;
   XiePhototag src1;
   XiePhototag src2;
   XieConstant src_constant;
   XiePhototag alpha;
   double alpha_const;
   XieProcessDomain *domain;
   unsigned int band_mask;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src1	Specifies the first data source.
src2	Specifies the second data source.
src_constant	Specifies the constant data source.
alpha	Specifies the blend proportion for each processed pixel.
alpha_const	Specifies the constant blend proportion for each processed pixel.
domain	Specifies the subset of source data that will be operated on.
band_mask	Specifies which bands are to be operated on.

Description

A Blend element produces output data from two data sources or a single data source and a constant. Each output pixel is a percentage combination of the source values, as controlled by the alpha input or the alpha constant.

When two sources are involved, *src1* and *src2* are the *phototags* of the elements supplying source data; *constant* is ignored. A phototag is the position or index of an element within an array of elements used to specify a photoflo; the first element in the array has a phototag value of one (1). If the operation is to involve a constant, *src1* is one operand, *src2* must be zero, and *constant* is used as the other operand. If *alpha* is non-zero, it controls the blend proportion for each pixel that is processed, otherwise *alpha_const* provides this control. *Domain* may control the subset of source data that will be operated upon. Only bands selected by the *band_mask* are subject to processing. Other bands present in the image are passed through to the output. For example, a *band_mask* of 001_2 indicates that only the "least significant band" would be processed; operating on all bands requires a *band_mask* of 111_2. Using *band_mask* to select source data that have two (2) or less levels is not permitted.

When two sources are involved, all attributes, other that *width* and *height*, must match. If *alpha* is non-zero, it must be a source of constrained data.

In order to specify a subset of source data that will be operated on, the `phototag`, `offset_x`, and `offset_y` fields of the `XieProcessDomain` structure pointed to by *domain* must be supplied; XIElib does not provide a convenience function to create and/or fill in an `XieProcessDomain` structure. If the entire source data is to be operated on, a pointer to an `XieProcessDomain` structure must still be provided, with the `phototag` field set to zero (0); the `offset_x` and `offset_y` fields are ignored.

Within the intersection of the source(s) and *`domain`*, each output pixel is a blend of the corresponding pixels from *`src1`* and *`src2`* (or *`src1`* pixels blended with *`constant`*). The degree of blend is determined by the corresponding value taken from *`alpha`* or the value of *`alpha_const`*. If *`alpha`* is non-zero, the proportion of blend is further scaled by *`alpha_const`*:

output = *src1* * (1 - *alpha* / *alpha_const*) + *src2* * (*alpha* / *alpha_const*)
(where *`alpha_const`* is greater than 0.0)

if *`alpha`* is zero:

output = *src1* * (1 - *alpha_const*) + *src2* * *alpha_const*
(where *`alpha_const`* is in the range [0.0, 1.0])

Output Attributes

Class	same as *`src1`*
Type	same as *`src1`*
Width	same as *`src1`*
Height	same as *`src1`*
Levels	same as *`src1`*

Structures

`XieFloBlend()` sets the `XiePhotoElement` structure field `elemType` to `xieElemBlend`, which identifies the element as a Blend, and sets the fields of the member structure `Blend` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
typedef float XieConstant[3];
typedef struct {
    int offset_x;
    int offset_y;
    XiePhototag phototag;
} XieProcessDomain;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src1;
         XiePhototag src2;
         XieConstant src_constant;
         XiePhototag alpha;
         float alpha_constant;
         XieProcessDomain domain;
         unsigned int band_mask;
      } Blend;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	Incompatible attributes between *`src1`* and *`src2`* *or* *`alpha`* is unconstrained *or* selected source data are bitonal
`xieErrNoFloSource`	Invalid *`src1`*, *`src2`*, or *`alpha`*
`xieErrNoFloValue`	*`alpha`* is zero and *`alpha_const`* is outside the range [0.0,1.0] *or* *`alpha`* is non-zero and *`alpha_const`* is non-positive

Name

XieFloCompare - specify a Compare element and set its parameters.

Synopsis

```
void XieFloCompare (element, src1, src2, domain, constant, operator,
            combine, band_mask)
   XiePhotoElement *element;
   XiePhototag src1;
   XiePhototag src2;
   XieProcessDomain *domain;
   XieConstant constant;
   XieCompareOp operator;
   Bool combine;
   unsigned int band_mask;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src1	Specifies the first data source.
src2	Specifies the second data source.
domain	Specifies the subset of source data that will be operated on.
constant	Specifies the constant data source.
operator	Specifies the logical predicate operator used in the comparison.
combine	Specifies if the comparison should be done on a per-band or on an all-bands basis.
band_mask	Specifies which bands are to be operated upon.

Description

A Compare element takes two data sources or a single data source and a constant and generates a Boolean bitmap output which reflects the results of a point-wise comparison. The output data has a value of 1 wherever the comparison is true, and a value of 0 everywhere else (i.e., comparison false or comparison not performed). The comparison may be performed on a per-band basis, or for all bands taken together.

When two sources are involved, *src1* and *src2* are the *phototags* of the elements supplying source data; *constant* is ignored. A phototag is the position or index of an element within an array of elements used to specify a photoflo; the first element in the array has a phototag value of one (1). If the operation is to involve a constant, *src1* is one operand, *src2* must be zero, and *constant* is used as the other operand.

In order to specify a subset of source data that will be operated on, the `phototag`, `offset_x`, and `offset_y` fields of the `XieProcessDomain` structure pointed to by *domain* must be supplied; XIElib does not provide a convenience function to create and/or fill in an `XieProcessDomain` structure. If the entire source data is to be operated on, a pointer to an `XieProcessDomain` structure must still be provided, with the `phototag` field set to zero (0); the `offset_x` and `offset_y` fields are ignored.

operator is the logical predicate operator used in the comparison. The valid operators for the Compare process element are:

Operator	src1 (operator) src2	src1 (operator) constant
`xieValLT`	src1 < src2	src1 < constant
`xieValLE`	src1 ≤ src2	src1 ≤ constant
`xieValEQ`	src1 = src2	src1 = constant
`xieValNE`	src1 ≠ src2	src1 ≠ constant
`xieValGT`	src1 > src2	src1 > constant
`xieValGE`	src1 ≥ src2	src1 ≥ constant

combine is a Boolean which determines if the comparison should be done on a per-band basis, or on an all bands basis. Only bands selected by *band_mask* are subject to processing. Other bands present in the image are passed through to the output. For example, a *band_mask* of 001_2 indicates that only the "least significant band" would be processed; operating on all bands requires a *band_mask* of 111_2.

If *combine* is `True` or *src1* is single band, the output data will form a single Boolean bitmap. If *src1* is triple band and *combine* is `False`, the output data will yield three separate Boolean bitmaps (for this case *band_mask* must specify all bands).

If *src1* is triple band and *combine* is `True`, only the EQ and NE operators are allowed; equality is established for each band selected by *band_mask*, and then the result is logically ANDed to derive equality (inequality is a logical NOT of this result). For this case, width and height must match for all bands selected by *band_mask*.

The relationship between *combine* and data class dependencies are given in the following table:

combine	input class	***band_mask***	output class
`True`	single band	n/a	single band
	triple band	selected bands	single band
`False`	single band	n/a	single band
	triple band	all bands	triple band

Output Attributes

Class	see Description
Type	constrained
Width	same as *src1*
Height	same as *src1*
Levels	2 per band (see Description)

Structures

`XieFloCompare()` sets the `XiePhotoElement` structure field `elemType` to `xieElemCompare`, which identifies the element as a Compare, and sets the fields of the member structure `Compare` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
typedef float XieConstant[3];
typedef struct {
    int offset_x;
    int offset_y;
    XiePhototag phototag;
} XieProcessDomain;
```

```
typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src1;
         XiePhototag src2;
         XieProcessDomain domain;
         XieConstant constant;
         XieCompareOp operator;
         Bool combine;
         unsigned int band_mask;
      } Compare;
      ...
   } data;
} XiePhotoElement;

/* Definitions of Compare Operators */
#define xieValLT                        1
#define xieValLE                        2
#define xieValEQ                        3
#define xieValNE                        4
#define xieValGT                        5
#define xieValGE                        6
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDomain`	Invalid *`domain`*
`xieErrNoFloMatch`	Class differs between *`src1`* and *`src2`* *or* invalid combination of *`operator`* and *`combine`* *or* triple band, and *`combine`* is false, and *`band_mask`* incomplete
`xieErrNoFloOperator`	Invalid *`operator`*
`xieErrNoFloSource`	Invalid *`src1`* or *`src2`*

Name

XieFloConstrain - specify a Constrain element and set its parameters.

Synopsis

```
void XieFloConstrain (element, src, levels, constrain_tech,
            constrain_param)
   XiePhotoElement *element;
   XiePhototag src;
   XieLevels levels;
   XieConstrainTechnique constrain_tech;
   XiePointer constrain_param;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
levels	Specifies the number of quantization levels desired in the output data.
constrain_tech	Specifies the technique to be used when constraining the data.
constrain_param	Specifies the list of additional parameters required by *constrain*.

Description

A Constrain element applies a quantization model to the image data to convert the data to a fixed number of quantization levels. Application of the quantization model may involve steps such as range shifting, scaling, clipping, and rounding.

src is the *phototag* of the element supplying source data. A phototag is the position or index of an element within an array of elements used to specify a photoflo; the first element in the array has a phototag value of one (1). *Level*s is the number of quantization levels desired in the output data. *constrain_tech* specifies the constrain technique to be used when constraining the data. *constrain_param* is the list of additional parameters required by *constrain_tech*.

If the input image is already constrained the data will be reconstrained.

One of the following standard constrain technique values can be assigned to *constrain_tech* :

```
xieValConstrainClipScale
xieValConstrainHardClip
```

If a vendor defined additional private constrain techniques, the values given to these techniques can be assigned to *constrain_tech*.

Output Attributes

Class	same as *src*
Type	constrained
Width	same as *src*
Height	same as *src*
Levels	*levels*

Structures

`XieFloConstrain()` sets the `XiePhotoElement` structure field `elemType` to `xieElemConstrain`, which identifies the element as a Constrain, and sets the fields of the member structure `Constrain` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
typedef unsigned long XieLevels[3];
typedef unsigned XieConstrainTechnique;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieLevels levels;
         XieConstrainTechnique constrain_tech;
         XiePointer constrain_param;
      } Constrain;
      ...
   } data;
} XiePhotoElement;

/* Definitions for ConstrainTechniques */
#define xieValConstrainClipScale       2
#define xieValConstrainHardClip        4
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloSource`	Invalid *src*
`xieErrNoFloTechnique`	Invalid *constrain_tech* or *constrain_param*

See Also

XieTecClipScale().

Name

XieFloConvertFromIndex - specify a ConvertFromIndex element and set its parameters.

Synopsis

```
void XieFloConvertFromIndex (element, src, colormap, data_class,
            precision)
   XiePhotoElement *element;
   XiePhototag src;
   Colormap colormap;
   XieDataClass data_class;
   unsigned int precision;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
colormap	Specifies the Colormap from which to obtain output pixel values.
data_class	Specifies if the output data is single band or triple band.
precision	Specifies the number of bits to be used from *colormap*'s RGB values.

Description

A ConvertFromIndex element converts Colormap index data into achromatic or trichromatic data.

data_class specifies if the output data is single band or triple band and can be set to one of the following standard data class values:

xieValSingleBand
xieValTripleBand

If *data_class* is single band and a trichromatic *colormap* is specified (static color, pseudo color, true color, or direct color), the gray shade for each pixel is taken from the red values in *colormap*. If *data_class* is triple band and an achromatic *colormap* is specified (static gray or gray scale), the red band will be replicated to populate the green and blue output bands.

The depth of *colormap* must match the Levels attribute of *src* (i.e., 2^{depth} must equal Levels).

Output Attributes

Class	*data_class*
Type	constrained
Width	same as *src*
Height	same as *src*
Levels	$2^{precision}$ (per band)

Structures

`XieFloConvertFromIndex()` sets the `XiePhotoElement` structure field `elemType` to `xieElemConvertFromIndex`, which identifies the element as a ConvertFromIndex, and sets the fields of the member structure `ConvertFromIndex` using the arguments in the argument list.

```
typedef unsigned XieDataClass;
typedef unsigned XiePhototag;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         Colormap colormap;
         XieDataClass data_class;
         unsigned int precision;
      } ConvertFromIndex;
      ...
   } data;
} XiePhotoElement;

/* Definitions of DataClass */
#define xieValSingleBand              1
#define xieValTripleBand              2
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloColormap`	Invalid *colormap*
`xieErrNoFloMatch`	Levels of *src* do not match depth of *colormap*
`xieErrNoFloSource`	Invalid *src*
`xieErrNoFloValue`	Invalid *data_class* or *precision*

Name

XieFloConvertFromRGB - specify a ConvertFromRGB element and set its parameters.

Synopsis

```
void XieFloConvertFromRGB (element, src, color_space, color_param)
   XiePhotoElement *element;
   XiePhototag src;
   XieColorspace color_space;
   XiePointer color_param;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data (RGB assumed).
color_space	Specifies the technique to be used in the conversion.
color_param	Specifies the list of additional parameters required by *color_space*.

Description

A ConvertFromRGB element converts RGB source data to an alternate colorspace.

The source data must be triple band, and all bands must have equal dimensions. The type and levels of the output data are determined by the *color_space*'s technique parameters. All other attributes are inherited from *src*.

ConvertFromRGB techniques define the trichromatic colorspaces known to a ConvertFromRGB element. One of the following standard ConvertFromRGB technique values can be assigned to *color_space*:

```
xieValRGBToCIELab
xieValRGBToCIEXYZ
xieValRGBToYCbCr
xieValRGBToYCC
```

If a vendor defined additional private ConvertFromRGB techniques, the private technique values given to these techniques can be assigned to *color_space*.

Output Attributes

Class	triple band
Type	*color_space* dependent
Width	same as *src*
Height	same as *src*
Levels	*color_space* dependent

Structures

`XieFloConvertFromRGB()` sets the `XiePhotoElement` structure field `elemType` to `xieElemConvertFromRGB`, which identifies the element as a ConvertFromRGB, and sets the fields of the member structure `ConvertFromRGB` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
```
typedef unsigned XieColorspace;

```
typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieColorspace color_space;
         XiePointer color_param;
      } ConvertFromRGB;
      ...
   } data;
} XiePhotoElement;

/* Definitions for Colorspace Conversions */
#define xieValRGBToCIELab              2
#define xieValRGBToCIEXYZ              4
#define xieValRGBToYCbCr               6
#define xieValRGBToYCC                 8
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	*`src`* is not triple band *or* unequal inter-band dimensions
`xieErrNoFloSource`	Invalid *`src`*
`xieErrNoFloTechnique`	Invalid *`color_space`* or *`color_param`*

See Also

XieTecRGBToCIELab(), XieTecRGBToCIEXYZ(), XieTecRGBToYCbCr XieTecRGBToYCC().

Name

XieFloConvertToIndex - specify a ConvertToIndex element and set its parameters.

Synopsis

```
void XieFloConvertToIndex (element, src, colormap, color_list,
            notify, color_alloc_tech, color_alloc_param)
   XiePhotoElement *element;
   XiePhototag src;
   Colormap colormap;
   XieColorList color_list;
   Bool notify;
   XieColorAllocTechnique color_alloc_tech;
   XiePointer color_alloc_param;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying the constrained source data
colormap	Specifies the Colormap from which to obtain output pixel values.
color_list	Specifies the list where Colormap indices are to be stored.
notify	Specifies whether to enable sending `ColorAlloc` events.
color_alloc_tech	Specifies the desired color allocation technique.
color_alloc_param	Specifies the list of additional parameters required by *color_alloc_tech*.

Description

A ConvertToIndex element allocates and/or matches colors or gray shades, as required, from a Colormap. It produces pixel indices as output data, and records indices that it allocates in *color_list*.

The specified *color_alloc_tech* technique may generate a `ColorAlloc` event to warn the client that results are of lesser fidelity than desired. *Notify* allows the client to be notified about inferior results from color allocation or matching.

src is the *phototag* of the element supplying constrained source data. A phototag is the position or index of an element within an array of elements used to specify a photoflo; the first element in the array has a phototag value of one (1). *Colormap* is the Colormap from which colors or gray shades are allocated and/or matched. *Color_list* is the list where allocated Colormap indices are to be stored. *color_alloc_tech* specifies the desired color allocation technique. *color_alloc_params* is the list of additional parameters required by *color_alloc_tech*.

color_list is purged of any colors it already contains when photoflo execution begins. Allocated Colormap indices can be freed using `XiePurgeColorList()`, `XieDestroyColorList()`, or by making *color_list* the target of an active photoflo.

ColorAlloc techniques define the recognized color allocation techniques used by the ConvertToIndex element. One of the following standard ColorAlloc technique values can be assigned to *color_alloc_tech*:

```
xieValColorAllocDefault
xieValColorAllocAll
xieValColorAllocMatch
xieValColorAllocRequantize
```

If a vendor defined additional private ColorAlloc techniques, the private technique values given to these techniques can be assigned to `color_alloc_tech`.

The server is required to support the default technique which is bound to one of the standard techniques defined above or a private technique.

Output Attributes

Class	single band
Type	constrained
Width	same as *src*
Height	same as *src*
Levels	2^{depth} (i.e. *colormap* depth)

Structures

`XieFloConvertToIndex()` sets the `XiePhotoElement` structure field `elemType` to `xieElemConvertToIndex`, which identifies the element as a ConvertToIndex, and sets the fields of the member structure `ConvertToIndex` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
typedef XID XieColorList;
typedef unsigned XieColorAllocTechnique;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         Colormap colormap;
         XieColorList color_list;
         Bool notify;
         XieColorAllocTechnique color_alloc_tech;
         XiePointer color_alloc_param;
      } ConvertToIndex;
      ...
   } data;
} XiePhotoElement;

/* Definitions for ColorAlloc Techniques */
#define xieValColorAllocDefault        0
#define xieValColorAllocAll            2
#define xieValColorAllocMatch          4
#define xieValColorAllocRequantize     6
```

Errors

`xieErrNoFloAccess`	*color_list* already being used by another active photoflo
`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloColorlist`	Invalid *color_list*
`xieErrNoFloColormap`	Invalid *colormap*
`xieErrNoFloMatch`	Unequal inter-band dimensions

`xieErrNoFloSource`	Invalid *src*
`xieErrNoFloTechnique`	Invalid *color_alloc_tech* or *color_alloc_param*

See Also

XiePurgeColorList(), XieDestroyColorList().
XieTecColorAllocAll(), XieTecColorAllocMatch(), XieTecColorAllocRequantize().

Name

XieFloConvertToRGB - specify a ConvertToRGB element and set its parameters.

Synopsis

```
void XieFloConvertToRGB (element, src, color_space, color_param)
   XiePhotoElement *element;
   XiePhototag src;
   XieColorspace color_space;
   XiePointer color_param;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
color_space	Specifies the technique that will be used for the conversion.
color_param	Specifies the list of additional parameters required by *convert_space*.

Description

A ConvertToRGB element converts alternate colorspace source data into RGB data.

The source data must be triple band, and all bands must have equal dimensions. The type and levels of the output data are determined by the *color_space*'s technique parameters. All other attributes are inherited from *src*.

ConvertToRGB techniques define the trichromatic colorspaces known to a ConvertToRGB element. One of the following standard ConvertFromRGB technique values can be assigned to *color_space*:

```
xieValCIELabToRGB
xieValCIEXYZToRGB
xieValYCbCrToRGB
xieValYCCToRGB
```

If a vendor defined additional private ConvertToRGB techniques, the private technique values given to these techniques can be assigned to *color_space*.

Output Attributes

Class	triple band
Type	*color_space* dependent
Width	same as *src*
Height	same as *src*
Levels	*color_space* dependent

Structures

`XieFloConvertToRGB()` sets the `XiePhotoElement` structure field `elemType` to `xieElemConvertToRGB`, which identifies the element as a ConvertToRGB, and sets the fields of the member structure `ConvertToRGB` using the arguments in the argument list.

typedef unsigned XieColorspace;
```
typedef unsigned XiePhototag;
```

```
typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieColorspace color_space;
         XiePointer color_param;
      } ConvertToRGB;
      ...
   } data;
} XiePhotoElement;

/* Definitions for Colorspace Conversions */
#define xieValCIELabToRGB              2
#define xieValCIEXYZToRGB              4
#define xieValYCbCrToRGB               6
#define xieValYCCToRGB                 8
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	*`src`* is not triple band *or* unequal inter-band dimensions
`xieErrNoFloSource`	Invalid *`src`*
`xieErrNoFloTechnique`	Invalid *`color_space`* or *`color_param`*

See Also

XieTecCIELabToRGB(), XieTecCIEXYZToRGB(), XieTecYCbCrToRGB(), XieTecYCCToRGB().

Name

XieFloConvolve - specify a Convolve element and set its parameters.

Synopsis

```
void XieFloConvolve (element, src, domain, kernel, kernel_size,
            band_mask, convolve_tech, convolve_param)
   XiePhotoElement *element;
   XiePhototag src;
   XieProcessDomain *domain;
   float *kernel;
   int kernel_size;
   unsigned int band_mask;
   XieConvolveTechnique convolve_tech;
   XiePointer convolve_param;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
domain	Specifies the subset of the image that will be operated upon.
kernel	Specifies the coefficients used in the convolution process
kernel_size	Specifies the dimension of *kernel*.
band_mask	Specifies which bands are to be operated upon.
convolve_tech	Specifies the technique for handling edge conditions.
convolve_param	Specifies the list of additional parameters required by *convolve_tech*.

Description

A Convolve element produces output data by convolving each input pixel value (and surrounding area) with the specified convolution kernel.

In order to specify a subset of source data that will be operated on, the `phototag`, `offset_x`, and `offset_y` fields of the `XieProcessDomain` structure pointed to by *domain* must be supplied; XIElib does not provide a convenience function to create and/or fill in an `XieProcessDomain` structure. If the entire source data is to be operated on, a pointer to an `XieProcessDomain` structure must still be provided, with the `phototag` field set to zero (0); the `offset_x` and `offset_y` fields are ignored.

kernel represents a square array of float data which has odd dimensions. Thus, a single dimension is used to specify *kernel_size*.

Only bands selected by the *band_mask* are subject to processing. Other bands present in the image are passed through to the output. For example, a *band_mask* of 001_2 indicates that only the “least significant band” would be processed; operating on all bands requires a *band_mask* of 111_2. Using *band_mask* to select source data that have two (2) or less levels is not permitted.

All output data attributes are inherited from the source data.

Convolve techniques provide various methods of handling edge conditions. These techniques determine what pixel values are used when Convolve requires data beyond the image bounds. One of the following standard convolve technique values can be assigned to *convolve_tech*:

```
xieValConvolveDefault
xieValConvolveConstant
xieValConvolveReplicate
```

If a vendor defined additional private convolve techniques, the private technique values given to these techniques can be assigned to *convolve_tech*.

The server is required to support the default technique which is bound to one of the standard techniques defined above or a private technique.

Output Attributes

Class	same as *src*
Type	same as *src*
Width	same as *src*
Height	same as *src*
Levels	same as *src*

Structures

`XieFloConvolve()` sets the `XiePhotoElement` structure field `elemType` to `xieElemConvolve`, which identifies the element as a Convolve, and sets the fields of the member structure `Convolve` using the arguments in the argument list.

```
typedef unsigned XieConvolveTechnique;
typedef unsigned XiePhototag;
typedef struct {
    int offset_x;
    int offset_y;
    XiePhototag phototag;
} XieProcessDomain;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieProcessDomain domain;
         float *kernel;
         int kernel_size;
         unsigned int band_mask;
         XieConvolveTechnique convolve_tech;
         XiePointer convolve_param;
      } Convolve;
      ...
   } data;
} XiePhotoElement;

/* Definitions for ConvolveTechniques */
#define xieValConvolveDefault          0
#define xieValConvolveConstant         2
#define xieValConvolveReplicate        4
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDomain`	Invalid *`domain`*
`xieErrNoFloMatch`	Selected source data are bitonal
`xieErrNoFloSource`	Invalid *`src`*
`xieErrNoFloTechnique`	Invalid *`convolve_tech`* or *`convolve_param`*
`xieErrNoFloValue`	Invalid *`kernel_size`* (e.g., not odd)

See Also

XieTecConvolveConstant().

Name

XieFloDither - specify a Dither element and set its parameters.

Synopsis

```
void XieFloDither (element, src, band_mask, levels, dither_tech,
            dither_param)
   XiePhotoElement *element;
   XiePhototag src;
   unsigned int band_mask;
   XieLevels levels;
   XieDitherTechnique dither_tech;
   XiePointer dither_param;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
band_mask	Specifies which bands are to be operated upon.
levels	Specifies the number of levels desired in the output data.
dither_tech	Specifies the desired dither technique.
dither_param	Specifies the list of additional parameters required by *dither_tech*.

Description

The Dither element is used to reduce the number of quantization levels in an image. It accomplishes this by affecting adjacent pixels (area affect) to make up for the lack of depth resolution.

Only bands selected by the *band_mask* are subject to processing. Other bands present in the image are passed through to the output. For example, a *band_mask* of 001_2 indicates that only the “least significant band” would be processed; operating on all bands requires a *band_mask* of 111_2. Using *band_mask* to select source data that have two (2) or less levels is not permitted.

The source data must be constrained.

Dither techniques define the technique that can be used to dither an image. One of the following standard dither technique values can be assigned to *dither_tech*:

```
xieValDitherDefault
xieValDitherErrorDiffusion
xieValDitherOrdered
```

If a vendor defined additional private dither techniques, the private technique values given to these techniques can be assigned to *dither_tech*.

The server is required to support the default technique which is bound to one of the standard techniques defined above or a private technique.

Output Attributes

Class	same as *src*
Type	constrained
Width	same as *src*

Height	same as *src*
Levels	*levels*

Structures

`XieFloDither()` sets the `XiePhotoElement` structure field `elemType` to `xieElemDither`, which identifies the element as a Dither, and sets the fields of the member structure `Dither` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
typedef unsigned long XieLevels[3];
typedef unsigned XieDitherTechnique;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieLevels levels;
         unsigned int band_mask;
         XieDitherTechnique dither_tech;
         XiePointer dither_param;
      } Dither;
      ...
   } data;
} XiePhotoElement;

/* Definitions for DitherTechniques */
#define xieValDitherDefault            0
#define xieValDitherErrorDiffusion     2
#define xieValDitherOrdered            4
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDomain`	Invalid *domain*
`xieErrNoFloMatch`	Unconstrained *src* data *or* selected source data are bitonal
`xieErrNoFloSource`	Invalid *src*
`xieErrNoFloTechnique`	Invalid *dither_tech* or *dither_param*
`xieErrNoFloValue`	Invalid output *levels*: less than two or greater than *src* levels

See Also

XieTecDitherOrderedParam().

Name

XieFloGeometry - specify a Geometry element and set its parameters.

Synopsis

```
void XieFloGeometry (element, src, width, height, coefficients,
            constant, band_mask, sample_tech, sample_param)
   XiePhotoElement *element;
   XiePhototag src;
   unsigned int width;
   unsigned int height;
   float coefficients[6];
   XieConstant constant;
   unsigned int band_mask;
   XieGeometryTechnique sample_tech;
   XiePointer sample_param;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
width	Specifies the horizontal dimension of the output data.
height	Specifies the vertical dimension of the output data.
coefficients	Specifies the list of values that control the geometric transformation.
constant	Specifies the fill value used for output pixels that do not map back to a *src* pixel.
band_mask	Specifies which bands are to be operated upon.
sample_tech	Specifies the technique to be used for retrospectively resampling *src*.
sample_param	Specifies the list of additional parameters required by *sample_tech*.

Description

A Geometry element is used to perform geometric transformations on image data. Linear geometric resampling operations are implemented, such as: crop, mirror, scale, shear, rotate, translate, and combinations thereof.

A Geometry element can be visualized as stepping through each possible output pixel location in turn, and computing the location from which to obtain each input pixel value. Each pixel (x',y') in the output image is mapped to the coordinate location (x,y) in *src* by:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

The coordinate mapping *coefficients* (a,b,c,d,t_x,t_y), together with the output *width* and *height*, fully specify the geometric transformation. The following briefly (and approximately) summarizes the intuitive role of each parameter:

a, d Scaling parameters. Increasing a and d will make the output image appear smaller, whereas decreasing them will make the output pixels appear larger.

b, c Rotation/skew parameters. If b and c are zero, the output image will be a rectangular scaling of the input image. If a and d are both zero, b is one and c is negative one, the image will appear rotated. The magnitude of b and c will affect scaling as well, if a and d are zero. If more than two of (a,b,c,d) are non-zero, the effect is complicated. The image may appear skewed and scaled.

t_x, ty Translation parameters. Used to specify the offset between input and output coordinate systems.

width, height These specify the output image dimensions of the selected band(s). Note that increasing the output image height and width over the input image size will not by itself cause magnification — if a and d are one (1) and b and c are zero (0), the output image will have the same appearance as the input, except that the borders will shrink or expand (as determined by width and height) and new areas of the image will be filled with *constant*.

The region to be cropped in the input image is implicitly defined by the dimensions of the output image and the mapping from output to input coordinates. Depending on the size of the input and output images, the amount of scaling specified, and the amount of translation in the mapping, pixels in the output image may map off the edge of the input image and the *constant* value is used.

Trichromatic image bands can be operate individually, all together, or in any combination, using *band_mask*. Since applying the same (a,b,c,d,t_x,t_y) mapping to inputs with diverse sizes will specify different transformations, operating on all bands in unison (*band_mask* of 111_2) is most appropriate if the dimensions of all bands are equal.

Often a given output pixel location (x′,y′), will not correspond exactly to a single pixel in the input image. The *sample_tech* technique is used to determine how the input data will be interpolated to produce each output pixel value. One of the following standard geometry technique values can be assigned to *sample_tech*:

```
xieValGeomDefault
xieValGeomAntialias
xieValGeomAntialiasByArea
xieValGeomAntialiasByLPF
xieValGeomBilinearInterp
xieValGeomGaussian
xieValGeomNearestNeighbor
```

If a vendor defined additional private geometry techniques, the private technique values given to these techniques can be assigned to *sample_tech*.

The server is required to support the default technique which is bound to one of the standard techniques defined above or a private technique.

Output Attributes

Class	same as *src*
Type	same as *src*
Width	*width*
Height	*height*
Levels	same as *src*

Structures

`XieFloGeometry()` sets the `XiePhotoElement` structure field `elemType` to `xieElemGeometry`, which identifies the element as a Geometry, and sets the fields of the member structure `Geometry` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
typedef float XieConstant[3];
typedef unsigned XieGeometryTechnique;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         unsigned int width;
         unsigned int height;
         float coefficients[6];
         XieConstant constant;
         unsigned int band_mask;
         XieGeometryTechnique sample_tech;
         XiePointer sample_param;
      } Geometry;
      ...
   } data;
} XiePhotoElement;

/* Definitions for GeometryTechniques */
#define xieValGeomDefault              0
#define xieValGeomAntialias            2
#define xieValGeomAntialiasByArea      4
#define xieValGeomAntialiasByLPF       6
#define xieValGeomBilinearInterp       8
#define xieValGeomGaussian             10
#define xieValGeomNearestNeighbor      12
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloSource`	Invalid *src*
`xieErrNoFloTechnique`	Invalid *sample_tech* or *sample_param*
`xieErrNoFloValue`	Invalid *coefficients*

See Also

XieTecGeomAntialiasByArea(), XieTecGeomAntialiasByLowpass(), XieTecGeomGaussian(), XieTecGeomNearestNeighbor().

Name

XieFloLogical - specify a Logical element and set its parameters.

Synopsis

```
void XieFloLogical (element, src1, src2, domain, constant, operator,
            band_mask)
   XiePhotoElement *element;
   XiePhototag src1;
   XiePhototag src2;
   XieProcessDomain *domain;
   XieConstant constant;
   unsigned long operator;
   unsigned int band_mask;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src1	Specifies the first data source.
src2	Specifies the second data source.
domain	Specifies the subset of source data that will be operated on.
constant	Specifies the constant data source.
operator	Specifies the logical operator to be used.
band_mask	Specifies which bands are to be operated upon.

Description

A Logical element performs per-pixel bit-wise operations on a single data source, or between two data sources, or between a single data source and a constant.

When two sources are involved, *src1* and *src2* are the phototags of the elements supplying source data; *constant* is ignored. A phototag is the position or index of an element within an array of elements used to specify a photoflo; the first element in the array has a phototag value of one (1). If the operation is to involve a constant, *src1* is one operand, *src2* must be zero, and *constant* is used as the other operand.

In order to specify a subset of source data that will be operated on, the `phototag`, `offset_x`, and `offset_y` fields of the `XieProcessDomain` structure pointed to by *domain* must be supplied; XIElib does not provide a convenience function to create and/or fill in an `XieProcessDomain` structure. If the entire source data is to be operated on, a pointer to an `XieProcessDomain` structure must still be provided, with the `phototag` field set to zero (0); the `offset_x` and `offset_y` fields are ignored.

The value of *operator* matches the GC-function values defined by the core protocol specification for CreateGC. The output of a Logical element is determined by the number of data sources and *operator*:

GC-function	monadic operation	dyadic operation
Clear	0	0
And	constant AND src1	src2 AND src1
AndReverse	constant AND (NOT src1)	src2 AND (NOT src1)
Copy	constant	src2
AndInverted	(NOT constant) AND src1	(NOT src2) AND src1
NoOp	src1	src1
Xor	constant XOR src1	src2 XOR src1
Or	constant OR src1	src2 OR src1
Nor	(NOT constant) AND (NOT src1)	(NOT src2) AND (NOT src1)
Equiv	(NOT constant) XOR src1	(NOT src2) XOR src1
Invert	NOT src1	NOT src1
OrReverse	constant OR (NOT src1)	src2 OR (NOT src1)
CopyInverted	NOT constant	NOT src2
OrInverted	(NOT constant) OR src1	(NOT src2) OR src1
Nand	(NOT constant) OR (NOT src1)	(NOT src2) OR (NOT src1)
Set	1	1

Only bands selected by the *band_mask* are subject to processing. Other bands present in the image are passed through to the output. For example, a *band_mask* of 001_2 indicates that only the "least significant band" would be processed; operating on all bands requires a *band_mask* of 111_2.

Output Attributes

Class	same as *src1*
Type	constrained
Width	same as *src1*
Height	same as *src1*
Levels	same as *src1*

Structures

`XieFloLogical()` sets the `XiePhotoElement` structure field `elemType` to `xieElemLogical`, which identifies the element as a Logical, and sets the fields of the member structure `Logical` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
typedef float XieConstant[3];
typedef struct {
    int offset_x;
    int offset_y;
    XiePhototag phototag;
} XieProcessDomain;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src1;
         XiePhototag src2;
         XieProcessDomain domain;
```

```
            XieConstant constant;
            int operator;
            unsigned int band_mask;
        } Logical;
        ...
    } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDomain`	Invalid *domain*
`xieErrNoFloMatch`	Class or levels differ between *src1* and *src2* *or* levels is not a power of 2 *or* *src1* or *src2* in not constrained
`xieErrNoFloOperator`	Invalid *operator*
`xieErrNoFloSource`	Invalid *src1* or *src2*

Name

XieFloMatchHistogram - specify a MatchHistogram element and set its parameters.

Synopsis

```
void XieFloMatchHistogram (element, src, domain, shape, shape_param)
   XiePhotoElement *element;
   XiePhototag src;
   XieProcessDomain *domain;
   XieHistogramShape shape;
   XiePointer shape_param;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
domain	Specifies the subset of source data that will be operated on.
shape	Specifies the form of the desired output data histogram.
shape_param	Specifies the list of additional parameters required by *shape*.

Description

A MatchHistogram element produces output data which differs from the source data in terms of its pixel value distribution, or histogram. It allows the client to specify the desired state of the resulting data's histogram (algorithmic description of resulting histogram shape).

The source data must be constrained and single band, and it must have three (3) or more levels.

In order to specify a subset of source data that will be operated on, the `phototag`, `offset_x`, and `offset_y` fields of the `XieProcessDomain` structure pointed to by *domain* must be supplied; XIElib does not provide a convenience function to create and/or fill in an `XieProcessDomain` structure. Only data that intersects with the subset specified by *domain* is included in the histogram, and only that data will be affected in the result of the histogram matching operation: all other data will pass-through unchanged. If the entire source data is to be operated on, a pointer to an `XieProcessDomain` structure must still be provided, with the `phototag` field set to zero (0); the `offset_x` and `offset_y` fields are ignored.

HistogramShape defines the various match-histogram shape techniques that can be requested by a MatchHistogram element. One of the following standard match-histogram shape technique values can be assigned to *shape*:

```
xieValHistogramFlat
xieValHistogramGaussian
xieValHistogramHyperbolic
```

If a vendor defined additional private match-histogram shape techniques, the private technique values given to these techniques can be assigned to *shape*.

Output Attributes

Class	single band
Type	constrained
Width	same as *src*

Height	same as *src*
Levels	same as *src*

Structures

`XieFloMatchHistogram()` sets the `XiePhotoElement` structure field `elemType` to `xieElemMatchHistogram`, which identifies the element as a MatchHistogram, and sets the fields of the member structure `MatchHistogram` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
typedef unsigned XieHistogramShape;
typedef struct {
    int offset_x;
    int offset_y;
    XiePhototag phototag;
} XieProcessDomain;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieProcessDomain domain;
         XieHistogramShape shape;
         XiePointer shape_param;
      } MatchHistogram;
      ...
   } data;
} XiePhotoElement;

/* Definitions for GeometryTechniques */
#define xieValGeomDefault              0
#define xieValGeomAntialias            2
#define xieValGeomAntialiasByArea      4
#define xieValGeomAntialiasByLPF       6
#define xieValGeomBilinearInterp       8
#define xieValGeomGaussian             10
#define xieValGeomNearestNeighbor      12
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDomain`	Invalid *domain*
`xieErrNoFloMatch`	Invalid *src* data: unconstrained, triple band, or bitonal
`xieErrNoFloSource`	Invalid *src*
`xieErrNoFloTechnique`	Invalid *shape* or *shape_param*

See Also

XieTecHistogramGaussian(), XieTecHistogramHyperbolic().

Name

XieFloMath - specify a Math element and set its parameters.

Synopsis

```
void XieFloMath (element, src, domain, operator, band_mask)
   XiePhotoElement *element;
   XiePhototag src;
   XieProcessDomain *domain;
   XieMathOp operator;
   unsigned int band_mask;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
domain	Specifies the subset of source data that will be operated on.
operator	Specifies the mathematical operation to be applied.
band_mask	Specifies which bands are to be operated upon.

Description

A Math element applies a single operand mathematical operation to the source data on a point-wise basis.

In order to specify a subset of source data that will be operated on, the `phototag`, `offset_x`, and `offset_y` fields of the `XieProcessDomain` structure pointed to by *domain* must be supplied; XIElib does not provide a convenience function to create and/or fill in an `XieProcessDomain` structure. If the entire source data is to be operated on, a pointer to an `XieProcessDomain` structure must still be provided, with the `phototag` field set to zero (0); the `offset_x` and `offset_y` fields are ignored.

Pixel computations that would lead to errors will yield valid server-dependent values (e.g. the log of a constrained pixel value of zero might result in a value of zero). Only bands selected by the *band_mask* are subject to processing. Other bands present in the image are passed through to the output. For example, a *band_mask* of 001_2 indicates that only the “least significant band” would be processed; operating on all bands requires a *band_mask* of 111_2. Using *band_mask* to select source data that have two (2) or less levels is not permitted.

The following valid mathematical operations that can be invoked through the Math element:

Operator	**Meaning**
`xieValExp`	exponential
`xieValLn`	natural logarithm
`xieValLog2`	logarithm base 2
`xieValLog10`	logarithm base 10
`xieValSquare`	square
`xieValSqrt`	square root

All output data attributes are inherited from the source data.

Output Attributes

Class	same as *src*
Type	same as *src*
Width	same as *src*
Height	same as *src*
Levels	same as *src*

Structures

`XieFloMath()` sets the `XiePhotoElement` structure field `elemType` to `xieElemMath`, which identifies the element as a Math, and sets the fields of the member structure `Math` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
typedef unsigned XieMathOp;
typedef struct {
    int offset_x;
    int offset_y;
    XiePhototag phototag;
} XieProcessDomain;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieProcessDomain domain;
         XieMathOp operator;
         unsigned int band_mask;
      } Math;
      ...
   } data;
} XiePhotoElement;

/* Definitions of Math Operators */
#define xieValExp                     1
#define xieValLn                      2
#define xieValLog2                    3
#define xieValLog10                   4
#define xieValSquare                  5
#define xieValSqrt                    6
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDomain`	Invalid *domain*
`xieErrNoFloMatch`	Selected source data are bitonal
`xieErrNoFloSource`	Invalid *src*
`xieErrNoFloOperator`	Invalid *operator*

Name

XieFloPasteUp - specify a PasteUp element and set its parameters.

Synopsis

```
void XieFloPasteUp (element, width, height, constant, tiles,
            tile_count)
   XiePhotoElement *element;
   unsigned int width;
   unsigned int height;
   XieConstant constant;
   XieTile *tiles
   unsigned int tile_count;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
width	Specifies the full horizontal extent of the output data.
height	Specifies the full vertical extent of the output data.
constant	Specifies the fill value for output regions that are not tiled.
tiles	Specifies a list of tile descriptors.
tile_count	Specifies the number of tile descriptors in *tiles*.

Description

A PasteUp element is an N-input translate operation which outputs data constructed from various source data tiles or a constant value.

Each of the *tile*s specifies a `src` (the *phototag* of the element supplying source data), and the coordinates, `dst_x` and `dst_y`, where the tile belongs in the output data. A phototag is the position or index of an element within an array of elements used to specify a photoflo; the first element in the array has a phototag value of one (1).

Each region of the output data is defined by a tile's destination coordinates, `dst_x` and `dst_y`, and its `src` dimensions. For output regions where no tile provides input, the value of *constant* is used. If tiles overlap, a stacking order rule defines which pixel value will be output: the last tile involved in the overlap in the list of *tiles* will provide the output pixel.

At least one tile must be supplied. Except for *width* and *height*, all attributes of each source tile must match. In addition, for triple band input, inter-band dimensions within each *tiles* must match.

Output Attributes

Class	same as *tiles*
Type	same as *tiles*
Width	*width*
Height	*height*
Levels	same as *tiles*

Structures

`XieFloPasteUp()` sets the `XiePhotoElement` structure field `elemType` to `xieElemPasteUp`, which identifies the element as a PasteUp, and sets the fields of the member structure `PasteUp` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
typedef float XieConstant[3];
typedef struct {
   XiePhototag src;
   int dst_x;
   int dst_y;
} XieTile;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         unsigned int width;
         unsigned int height;
         XieConstant constant;
         XieTile *tiles;
         unsigned int tile_count;
      } PasteUp;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	Incompatible attributes between *`tiles`* *or* unequal inter-band dimensions within a tile
`xieErrNoFloSource`	Invalid source *`tiles`* *or* no *`tiles`* were specified

See Also

XieFreePasteUpTiles().

Name

XieFloPoint - specify a Point element and set its parameters.

Synopsis

```
void XieFloPoint (element, src, domain, lut, band_mask)
   XiePhotoElement *element;
   XiePhototag src;
   XieProcessDomain *domain;
   XieLut lut;
   unsigned int band_mask;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
domain	Specifies the subset of source data that will be operated on.
lut	Specifies the LUT resource supplying the lookup table.
band_mask	Specifies which bands are to be operated upon.

Description

A Point element maps source pixel values to output pixel values using a lookup table (LUT).

src is the *phototag* of the element supplying constrained source data. A phototag is the position or index of an element within an array of elements used to specify a photoflo; the first element in the array has a phototag value of one (1). *Lut* is the phototag of the element supplying the lookup table array(s).

In order to specify a subset of source data that will be operated on, the `phototag`, `offset_x`, and `offset_y` fields of the `XieProcessDomain` structure pointed to by *domain* must be supplied; XIElib does not provide a convenience function to create and/or fill in an `XieProcessDomain` structure. If the entire source data is to be operated on, a pointer to an `XieProcessDomain` structure must still be provided, with the `phototag` field set to zero (0); the `offset_x` and `offset_y` fields are ignored.

band_mask specifies which bands are to be operated upon (all bands must be specified if *lut* is single band and *src* is triple band). For example, a *band_mask* of 001_2 indicates that only the "least significant band" would be processed; operating on all bands requires a *band_mask* of 111_2.

The output is constrained, with the width and height taken from *src*, and class and levels taken from *lut*. When *src* is single band and *lut* is triple band, for the bands that are indicated by *band_mask*, the output bands are remapped through their respective *lut* bands, whereas the other bands are just replications of the single *src* band. If *domain* is used, the class and levels of *lut* must match those of *src*.

Each *lut* array must contain sufficient entries so that all potential pixel values found in *src* can form a valid index into the array. Generally each *src* pixel value is used directly as an index into a *lut* array. When triple band *src* data are remapped through a single band *lut* however, pixel values from all three *src* bands are combined to form an array index; for this case, width and height must match for all bands.

When one LUT array is used with triple band data, the following presents the algorithm for computing combined array indices:

LUT band-order	LUT indexing algorithm for combining pixel values
LSFirst	index = value[0] + value[1] x levels[0] + value[2] x levels[0] x levels[1]
MSFirst	index = value[2] + value[1] x levels[2] + value[0] x levels[2] x levels[1]

Output Attributes

Class	same as *lut*
Type	constrained
Width	same as *src*
Height	same as *src*
Levels	same as *lut*

Structures

`XieFloPoint()` sets the `XiePhotoElement` structure field `elemType` to `xieElemPoint`, which identifies the element as a Point, and sets the fields of the member structure `Point` using the arguments in the argument list.

```
typedef XID XieLut;
typedef unsigned XiePhototag;
typedef struct {
    int offset_x;
    int offset_y;
    XiePhototag phototag;
} XieProcessDomain;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieProcessDomain domain;
         XieLut lut;
         unsigned int band_mask;
      } Point;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDomain`	Invalid *domain*
`xieErrNoFloMatch`	Unconstrained *src* data *or* *lut* does not contain enough entries *or* *lut* is single band and src is triple band, but *band_mask* is incomplete *or* *domain* is being used, but *lut* class or levels do not match those of *src*
`xieErrNoFloSource`	Invalid *src* or *lut*

Name

XieFloUnconstrain - specify an unconstrain element and set its parameters.

Synopsis

```
void XieFloUnconstrain (element, src)
   XiePhotoElement *element;
   XiePhototag src;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying constrained source data.

Description

An Unconstrain element produces unconstrained output data from constrained input data.

Output Attributes

Class	same as *src*
Type	unconstrained
Width	same as *src*
Height	same as *src*
Levels	unknown

Structures

`XieFloUnconstrain()` sets the `XiePhotoElement` structure field `elemType` to `xieElemUnconstrain`, which identifies the element as an Unconstrain, and sets the fields of the member structure `Unconstrain` using the arguments in the argument list.

```
typedef unsigned XiePhototag;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
      } Unconstrain;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	Unconstrained *src* data
`xieErrNoFloSource`	Invalid *src*

Name

XieFloExportClientHistogram - specify an ExportClientHistogram element and set its parameters.

Synopsis

```
void XieFloExportClientHistogram (element, src, domain, notify)
   XiePhotoElement *element;
   XiePhototag src;
   XieProcessDomain *domain;
   XieExportNotify notify;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying single band constrained source data.
domain	Specifies the subset of the source data from which the distribution will be generated.
notify	Specifies whether to enable sending an `ExportAvailable` event.

Description

An ExportClientHistogram element generates a histogram of the pixel values found in the source data. It prepares histogram data that can be retrieved by the client using `XieGetClientData()`. An event can be requested that will notify the client when histogram data becomes available.

The data generated for the client is a list of `XieHistogramData` where each entry consists of a value (i.e., a pixel value) followed by the count of pixels found with that value. If the number of pixels for a given value exceeds the capacity of count, that count will be returned at the maximum value (i.e., 2^{32}-1). Pixel values that are not found in the data are not included in the histogram data: no entries are returned where count is zero.

In order to specify a subset of source data that will be operated on, the `phototag`, `offset_x`, and `offset_y` fields of the `XieProcessDomain` structure pointed to by *`domain`* must be supplied; XIElib does not provide a convenience function to create and/or fill in an `XieProcessDomain` structure. Only data that intersects with the subset specified by *`domain`* is included in the histogram. If the entire source data is to be operated on, a pointer to an `XieProcessDomain` structure must still be provided, with the `phototag` field set to zero (0); the `offset_x` and `offset_y` fields are ignored.

One of three standard export notify values can be assigned to *`notify`*:

```
xieValDisable
xieValFirstData
xieValNewData
```

If *`notify`* was specified as `xieValFirstData`, this event will only be sent the first time data become available; otherwise, if `xieValNewData` was specified, this event will be generated each time the amount of data available changes from zero to non-zero.

Structures

`XieFloExportClientHistogram()` sets the `XiePhotoElement` structure field `elemType` to `xieElemExportClientHistogram`, which identifies the element as an

ExportClientHistogram, and sets the fields of the member structure `ExportClientHistogram` using the arguments in the argument list.

```
typedef unsigned XieExportNotify;
typedef unsigned XiePhototag;
typedef struct {
    int offset_x;
    int offset_y;
    XiePhototag phototag;
} XieProcessDomain;
typedef struct {
   unsigned long value;
   unsigned long count;
} XieHistogramData;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieProcessDomain domain;
         XieExportNotify notify;
      } ExportClientHistogram;
      ...
   } data;
} XiePhotoElement;

/* Definitions of ExportNotify */
#define xieValDisable                  1
#define xieValFirstData                2
#define xieValNewData                  3
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDomain`	Invalid *`domain`*
`xieErrNoFloMatch`	Unconstrained *`src`* data *or* triple band *`src`* data
`xieErrNoFloSource`	Invalid *`src`*
`xieErrNoFloValue`	Invalid *`notify`*

See Also

XieGetClientData().

Name

XieFloExportClientLUT - specify an ExportClientLUT element and set its parameters.

Synopsis

```
void XieFloExportClientLUT (element, src, band_order, notify, start,
            length)
   XiePhotoElement *element;
   XiePhototag src;
   XieOrientation band_order;
   XieExportNotify notify;
   XieLTriplet start;
   XieLTriplet length;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying lookup table data.
band_order	Specifies the order of triple band arrays.
notify	Specifies whether to enable sending an `ExportAvailable` event.
start	Specifies the index of the first array entry that should be returned.
length	Specifies the number of array entries that should be returned.

Description

An ExportClientLUT element allows data imported from an ImportLUT or an ImportClientLUT element to be retrieved by the client. The actual transport of lookup table data through the protocol stream is requested using `XieGetClientData()`.

One of three standard export notify values can be assigned to *notify*:

```
xieValDisable
xieValFirstData
xieValNewData
```

If *notify* was specified as `xieValFirstData`, this event will only be sent the first time data become available; otherwise, if `xieValNewData` was specified, this event will be generated each time the amount of data available changes from zero to non-zero.

band_order is the order in which triple band arrays are transmitted through the protocol stream. One of the following standard orientation values can be assigned to *band_order*:

```
xieValLSFirst
xieValMSFirst
```

The least significant band of trichromatic data is the first band mentioned in the common name of the colorspace: red is the least significant band of RGB data. For band-by-plane data, *band_order* specifies whether this band corresponds with the least significant or most significant LUT array. Each array is transported as a separate data stream:

band	LSFirst	MSFirst
0	$R_7R_6R_5R_4R_3R_2R_1R_0$	$B_7B_6B_5B_4B_3B_2B_1B_0$
1	$G_7G_6G_5G_4G_3G_2G_1G_0$	$G_7G_6G_5G_4G_3G_2G_1G_0$
2	$B_7B_6B_5B_4B_3B_2B_1B_0$	$R_7R_6R_5R_4R_3R_2R_1R_0$

The size of each array entry is either 1, 2, or 4 bytes: the smallest size into which the output quantization levels can be stored. When array entries require multiple bytes, the byte order per entry is determined in the same manner as other numeric data: it is the byte orientation established at core X connection setup time.

Structures

`XieFloExportClientLUT()` sets the `XiePhotoElement` structure field `elemType` to `xieElemExportClientLUT`, which identifies the element as an ExportClientLUT, and sets the fields of the member structure `ExportClientLUT` using the arguments in the argument list.

```
typedef unsigned XieExportNotify;
typedef unsigned XieOrientation;
typedef unsigned long XieLTriplet[3];
typedef unsigned XiePhototag;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieOrientation band_order;
         XieExportNotify notify;
         XieLTriplet start;
         XieLTriplet length;
      } ExportClientLUT;
      ...
   } data;
} XiePhotoElement;

/* Definitions of ExportNotify */
#define xieValDisable                  1
#define xieValFirstData                2
#define xieValNewData                  3

/* Definitions of Orientation Types */
#define xieValLSFirst                  1
#define xieValMSFirst                  2
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	*`start`* + *`length`* exceeds number of entries in an array
`xieErrNoFloSource`	Invalid *`src`*
`xieErrNoFloValue`	Invalid *`notify`* or *`band_order`*

See Also

XieFloImportLUT(), XieFloImportClientLUT(), XieGetClientData().

Name

XieFloExportClientPhoto - specify an ExportClientPhoto element and set its parameters.

Synopsis

```
void XieFloExportClientPhoto (element, src, notify, encode_tech,
            encode_param)
   XiePhotoElement *element;
   XiePhototag src;
   XieExportNotify notify;
   XieEncodeTechnique encode_tech;
   XiePointer encode_param;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying constrained data.
notify	Specifies whether to enable sending an `ExportAvailable` event.
encode_tech	Specifies the technique to compress or format the exported data.
encode_param	Specifies the list of additional parameters required by *encode_tech*.

Description

An ExportClientPhoto element makes image data available to the protocol stream. The attributes of the exported data are determined by the attributes of the source data. The format of the data is specified by the *encode_tech* technique and *encode_param*. The actual transport of image data through the protocol stream is requested using `XieGetClientData()`.

One of three standard export notify values can be assigned to *notify*:

```
xieValDisable
xieValFirstData
xieValNewData
```

If *notify* was specified as `xieValFirstData`, this event will only be sent the first time data become available; otherwise, if `xieValNewData` was specified, this event will be generated each time the amount of data available changes from zero to non-zero.

Encode techniques define the techniques that can be used to compress an image or format it as uncompressed data. One of the following standard encode technique values can be assigned to *encode_tech*:

```
xieValEncodeServerChoice
xieValEncodeUncompressedSingle
xieValEncodeUncompressedTriple
xieValEncodeG31D
xieValEncodeG32D
xieValEncodeG42D
xieValEncodeJPEGBaseline
xieValEncodeJPEGLossless
xieValEncodeTIFF2
xieValEncodeTIFFPackBits
```

If a vendor defined additional private encode techniques, the private technique values given to these techniques can be assigned to *encode_tech*.

Structures

`XieFloExportClientPhoto()` sets the `XiePhotoElement` structure field `elemType` to `xieElemExportClientPhoto`, which identifies the element as an ExportClientPhoto, and sets the fields of the member structure `ExportClientPhoto` using the arguments in the argument list.

```
typedef unsigned XieExportNotify;
typedef unsigned XiePhototag;
typedef unsigned XieEncodeTechnique;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieExportNotify notify;
         XieEncodeTechnique encode_tech;
         XiePointer encode_param;
      } ExportClientPhoto;
      ...
   } data;
} XiePhotoElement;

/* Definitions of ExportNotify */
#define xieValDisable                  1
#define xieValFirstData                2
#define xieValNewData                  3

/* Definitions for EncodeTechniques */
#define xieValEncodeServerChoice       1
#define xieValEncodeUncompressedSingle 2
#define xieValEncodeUncompressedTriple 3
#define xieValEncodeG31D               4
#define xieValEncodeG32D               6
#define xieValEncodeG42D               8
#define xieValEncodeJPEGBaseline       10
#define xieValEncodeJPEGLossless       12
#define xieValEncodeTIFF2              14
#define xieValEncodeTIFFPackBits       16
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloMatch`	Unconstrained *src* data
`xieErrNoFloSource`	Invalid *src*
`xieErrNoFloTechnique`	Invalid *encode_tech* or *encode_param*
`xieErrNoFloValue`	Invalid *notify*

See Also

XieGetClientData().
XieTecEncodeUncompressedSingle(), XieTecEncodeUncompressedTriple(), XieTecEncodeG31D(), XieTecEncodeG32D(), XieTecEncodeG42D(), XieTecEncodeServerChoice(), XieTecEncodeJPEGBaseline(), XieTecEncodeJPEGLossless(), XieTecEncodeTIFF2(), XieTecEncodeTIFFPackBits().

Name

XieFloExportClientROI - specify an ExportClientROI element and set its parameters.

Synopsis

```
void XieFloExportClientROI (element, src, notify)
   XiePhotoElement *element;
   XiePhototag src;
   XieExportNotify notify;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying the list-of-rectangles.
notify	Specifies whether to enable sending an `ExportAvailable` event.

Description

An ExportClientROI element allows a list-of-rectangles, imported using an ImportROI or an ImportClientROI element, to be retrieved by the client. The actual transport of list-of-rectangles data through the protocol stream is requested using a GetClientData element.

One of three standard export notify values can be assigned to *notify*:

```
xieValDisable
xieValFirstData
xieValNewData
```

If *notify* was specified as `xieValFirstData`, this event will only be sent the first time data become available; otherwise, if `xieValNewData` was specified, this event will be generated each time the amount of data available changes from zero to non-zero.

Structures

`XieFloExportClientROI()` sets the `XiePhotoElement` structure field `elemType` to `xieElemExportClientROI`, which identifies the element as an ExportClientROI, and sets the fields of the member structure `ExportClientROI` using the arguments in the argument list.

```
typedef unsigned XieExportNotify;
typedef unsigned XiePhototag;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieExportNotify notify;
      } ExportClientROI;
      ...
   } data;
} XiePhotoElement;

/* Definitions of ExportNotify */
#define xieValDisable                  1
```

```
#define xieValFirstData                2
#define xieValNewData                  3
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloSource`	Invalid *`src`*
`xieErrNoFloValue`	Invalid *`notify`*

See Also

XieFloImportROI(), XieFloImportClientROI(), XieFloGetClientData().

Name

XieFloExportDrawable - specify an ExportDrawable element and set its parameters.

Synopsis

```
void XieFloExportDrawable (element, src, drawable, gc, dst_x, dst_y)
   XiePhotoElement *element;
   XiePhototag src;
   Drawable drawable;
   GC gc;
   int dst_x;
   int dst_y;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying constrained source data.
drawable	Specifies the Window or Pixmap into which the data will be written.
gc	Specifies the GContext to be used when transferring pixels to *drawable*.
dst_x	Specifies where the data should be placed in *drawable*.
dst_y	Specifies where the data should be placed in *drawable*.

Description

An ExportDrawable element allows Colormap index data to be exported to a Window or Pixmap.

The following components are used from *gc*: function, plane-mask, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask.

The levels of *src* must exactly match the depth of *drawable* and *gc* (i.e., levels must be 2^{depth}).

Structures

`XieFloExportDrawable()` sets the `XiePhotoElement` structure field `elemType` to `xieElemExportDrawable`, which identifies the element as an ExportDrawable, and sets the fields of the member structure `ExportDrawable` using the arguments in the argument list.

```
typedef unsigned XiePhototag;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         Drawable drawable;
         GC gc;
         int dst_x;
         int dst_y;
      } ExportDrawable;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDrawable`	Invalid *drawable*
`xieErrNoFloGC`	Invalid *gc*
`xieErrNoFloMatch`	Invalid *src* data: triple band, unconstrained, levels doesn't match depth
`xieErrNoFloSource`	Invalid *src*

Name

XieFloExportDrawablePlane - specify an ExportDrawablePlane element and set its parameters.

Synopsis

```
void XieFloExportDrawablePlane (element, src, drawable, gc, dst_x,
            dst_y)
   XiePhotoElement *element;
   XiePhototag src;
   Drawable drawable;
   GC gc;
   int dst_x;
   int dst_y;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying constrained bitonal source data.
drawable	Specifies the Window or Pixmap into which the data will be written.
gc	Specifies the GContext to be used when transferring pixels to *drawable*.
dst_x	Specifies where the data should be placed in *drawable*.
dst_y	Specifies where the data should be placed in *drawable*.

Description

An ExportDrawablePlane element allows single-band single-bit (bitonal) data to be exported to a Window or a Pixmap.

The following components are used from *gc*: function, plane-mask, foreground, background, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. For the fill-style component of *gc*, values of `FillSolid` and `FillTiled` are treated as synonyms for `FillOpaqueStippled`.

Structures

`XieFloExportDrawablePlane()` sets the `XiePhotoElement` structure field `elemType` to `xieElemExportDrawablePlane`, which identifies the element as an ExportDrawablePlane, and sets the fields of the member structure `ExportDrawablePlane` using the arguments in the argument list.

```
typedef unsigned XiePhototag;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         Drawable drawable;
         GC gc;
         int dst_x;
         int dst_y;
      } ExportDrawablePlane;
      ...
```

```
    } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloDrawable`	Invalid *`drawable`*
`xieErrNoFloGC`	Invalid *`gc`*
`xieErrNoFloMatch`	Invalid *`src`* data: triple band, not constrained, levels > 2
`xieErrNoFloSource`	Invalid *`src`*

Name

XieFloExportLUT - specify an ExportLUT element and set its parameters.

Synopsis

```
void XieFloExportLUT (element, src, lut, merge, start)
   XiePhotoElement *element;
   XiePhototag src;
   XieLut lut;
   Bool merge;
   XieLTriplet start;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying lookup table data.
lut	Specifies the ID of the LUT to receive the data.
merge	Specifies how new array entries replace existing entries.
start	Specifies the index of the first array entry that should be written in *lut*.

Description

An ExportLUT element allows data imported from an ImportLUT or ImportClientLUT element to be saved in an existing LUT resource.

merge specifies that new array entries from *src* should replace entries that already exist within *lut*. If *merge* is `False`, *start* must be 0 for each band. In this case, *lut* will inherit the attributes of *src* and be populated with its data; the previous attributes and data of *lut* are overwritten when the photoflo completes. If *merge* is `True` and *lut* has existing attributes, the data from *src* will replace the data from *lut*, beginning at position *start*. If *merge* is `True` but *lut* has not yet been populated, it is an error.

The attributes of *src* must match those of *lut*, and the combination of *start* and the length of *src* must specify a valid sub-range existing within *lut*.

Structures

`XieFloExportLUT()` sets the `XiePhotoElement` structure field `elemType` to `xieElemExportLUT`, which identifies the element as an ExportLUT, and sets the fields of the member structure `ExportLUT` using the arguments in the argument list.

```
typedef XID XieLut;
typedef unsigned long XieLTriplet[3];
typedef unsigned XiePhototag;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieLut lut;
         Bool merge;
```

```
            XieLTriplet start;
        } ExportLUT;
        ...
    } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloLUT`	Invalid *`lut`*
`xieErrNoFloMatch`	*`merge`* true and attributes do not match between *`src`* and *`lut`* *or* *`merge`* true and *`start`* + *`src`* length is not a sub-range of *`lut`*
`xieErrNoFloSource`	Invalid *`src`*
`xieErrNoFloValue`	*`merge`* false and *`start`* non-zero

See Also

XieFloImportLUT(), XieFloImportClientLUT().

Name

XieFloExportPhotomap - specify an ExportPhotomap element and set its parameters.

Synopsis

```
void XieFloExportPhotomap (element, src, photomap, encode_tech,
            encode_param)
   XiePhotoElement *element;
   XiePhototag src;
   XiePhotomap photomap;
   XieEncodeTechnique encode_tech;
   XiePointer encode_param;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying source data.
photomap	Specifies the photomap structure to receive the data.
encode_tech	Specifies the image compression or formatting technique.
encode_param	Specifies the list of additional parameters required by *encode_tech*.

Description

An ExportPhotomap element allows data from photoflo operations to be saved in a photomap. A photomap is a server resource which can be used to store image data.

photomap will inherit the attributes of *src* and be populated with its data. The previous attributes and data of *photomap* are overwritten when the photoflo completes.

Encode techniques define the techniques that can be used to compress an image or format it as uncompressed data. One of the following standard encode technique values can be assigned to *encode_tech*:

```
xieValEncodeServerChoice
xieValEncodeUncompressedSingle
xieValEncodeUncompressedTriple
xieValEncodeG31D
xieValEncodeG32D
xieValEncodeG42D
xieValEncodeJPEGBaseline
xieValEncodeJPEGLossless
xieValEncodeTIFF2
xieValEncodeTIFFPackBits
```

If a vendor defined additional private encode techniques, the private technique values given to these techniques can be assigned to *encode_tech*.

Structures

`XieFloExportPhotomap()` sets the `XiePhotoElement` structure field `elemType` to `xieElemExportPhotomap`, which identifies the element as an ExportPhotomap, and sets the fields of the member structure `ExportPhotomap` using the arguments in the argument list.

```
typedef unsigned XiePhototag;
```

```
typedef XID XiePhotomap;
typedef unsigned XieEncodeTechnique;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XiePhotomap photomap;
         XieEncodeTechnique encode_tech;
         XiePointer encode_param;
      } ExportPhotomap;
      ...
   } data;
} XiePhotoElement;

/* Definitions for EncodeTechniques */
#define xieValEncodeServerChoice       1
#define xieValEncodeUncompressedSingle 2
#define xieValEncodeUncompressedTriple 3
#define xieValEncodeG31D               4
#define xieValEncodeG32D               6
#define xieValEncodeG42D               8
#define xieValEncodeJPEGBaseline       10
#define xieValEncodeJPEGLossless       12
#define xieValEncodeTIFF2              14
#define xieValEncodeTIFFPackBits       16
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloPhotomap`	Invalid *photomap*
`xieErrNoFloSource`	Invalid *src*
`xieErrNoFloTechnique`	Invalid *encode_tech* or *encode_param*

See Also

XieTecEncodeUncompressedSingle(), XieTecEncodeUncompressedTriple(), XieTecEncodeG31D(), XieTecEncodeG32D(), XieTecEncodeG42D(), XieTecEncodeServerChoice(), XieTecEncodeJPEGBaseline(), XieTecEncodeTIFF2(), XieTecEncodeTIFFPackBits().

Name

XieFloExportROI - specify an ExportROI element and set its parameters.

Synopsis

```
void XieFloExportROI (element, src, roi)
   XiePhotoElement *element;
   XiePhototag src;
   XieRoi roi;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.
src	Specifies the element supplying a list-of-rectangles.
roi	Specifies the ID of the ROI to receive the data.

Description

An ExportROI element allows data imported from an ImportROI or ImportClientROI element to be saved in an existing Rectangles-Of-Interest (ROI).

roi will be populated with new data. The previous data of *roi* are overwritten after the photoflo completes.

Structures

`XieFloExportROI()` sets the `XiePhotoElement` structure field `elemType` to `xieElemExportROI`, which identifies the element as an ExportROI, and sets the fields of the member structure `ExportROI` using the arguments in the argument list.

```
typedef XID XieRoi;
typedef unsigned XiePhototag;

typedef struct {
   int elemType;
   union {
      ...
      struct {
         XiePhototag src;
         XieRoi roi;
      } ExportROI;
      ...
   } data;
} XiePhotoElement;
```

Errors

`xieErrNoFloAlloc`	Insufficient resources (e.g. memory)
`xieErrNoFloROI`	Invalid *roi*
`xieErrNoFloSource`	Invalid *src*

See Also

XieFloImportROI(), XieFloImportClientROI().

Name

XieTecColorAllocAll - allocate and fill an `XieColorAllocAllParam` structure.

Synopsis

```
XieColorAllocAllParam *XieTecColorAllocAll (fill)
   unsigned long fill;
```

Arguments

fill	Specifies the value to map new pixels.

Returns

The `XieColorAllocAllParam` structure.

Description

`XieTecColorAllocAll()` allocates and returns a pointer to an `XieColorAllocAllParam` structure. The returned structure represents the list of parameters required by the AllocAll color allocation technique and may be used as the *color_alloc_param* argument of `XieFloConvertToIndex()` (when the *color_alloc_tech* argument is `xieValColorAllocAll`).

If insufficient memory is available, `XieTecColorAllocAll()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The *AllocAll* color allocation technique allocates a read-only Colormap cell for each new pixel found. If the Colormap runs out of cells, the remaining new pixels are mapped to *fill*. A `ColorAlloc` event, which warns the client that results are of lesser fidelity than desired, can be sent if it is necessary to use *fill*. AllocAll is only appropriate for dynamic Colormaps and requires that the number of discrete image pixels fit within the size of the Colormap to avoid running out of cells.

Structures

`XieTecColorAllocAll()` sets the structure field `fill` to the value of the argument *fill*.

```
typedef struct {
   unsigned long fill;
} XieColorAllocAllParam;
```

See Also

XieFloConvertToIndex().

Name

XieTecColorAllocMatch - allocate and fill an `XieColorAllocMatchParam` structure.

Synopsis

```
XieColorAllocMatchParam *XieTecColorAllocMatch (match_limit,
            gray_limit)
   double match_limit;
   double gray_limit;
```

Arguments

match_limit	Specifies the color allocation control value.
gray_limit	Specifies the gray scale allocation control value.

Returns

The `XieColorAllocMatchParam` structure.

Description

`XieTecColorAllocMatch()` allocates and returns a pointer to an `XieColorAllocMatchParam` structure. The returned structure represents the list of parameters required by the AllocMatch color allocation technique and may be used as the *color_alloc_param* argument of `XieFloConvertToIndex()` (when the *color_alloc_tech* argument is `xieValColorAllocMatch`).

If insufficient memory is available, `XieTecColorAllocMatch()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The AllocMatch color allocation technique allows a trade-off between image fidelity and Colormap usage via a pair of granularity parameters. The highest priority is given to allocating read-only cells in a sequence that provides an even distribution of pixels throughout the colorspace. Secondary priority is given to the frequency of usage of image pixels. Any image pixel which is a close enough match to an existing read-only cell will share that cell (where "close" is determined by the granularity controls). For other image pixels, new read-only allocations are made. When no more cells are available, each remaining image pixel is matched to the closest read-only cell. The AllocMatch color allocation technique is appropriate for both static and dynamic Colormaps. For the sake of computational efficiency the number of discrete image pixels should not exceed the size of the Colormap.

match_limit and *gray_match* control the allocation of colors and gray shades respectively. The minimum value (0.0) specifies exact matches (within the limits of the Colormap). The maximum value (1.0) encompasses the entire colorspace within which no new cells are allocated. A `ColorAlloc` event, which warns the client that results are of lesser fidelity than desired, can be sent if the Colormap runs out of cells.

Structures

`XieTecColorAllocMatch()` sets the structure field `match_limit` to the value of the argument *match_limit*; and the structure field `gray_limit` to the value of the argument *gray_limit*.

```
typedef struct {
   float match_limit;
   float gray_limit;
} XieColorAllocMatchParam;
```

See Also

XieFloConvertToIndex().

Name

XieTecColorAllocRequantize - allocate and fill an `XieColorAllocRequantizeParam` structure.

Synopsis

```
XieColorAllocRequantizeParam *XieTecColorAllocRequantize (max_cells)
   unsigned long max_cells;
```

Arguments

max_cells Specifies the maximum number of Colormap allocations to allow.

Returns

The `XieColorAllocRequantizeParam` structure.

Description

`XieTecColorAllocRequantize()` allocates and returns a pointer to an `XieColorAllocRequantizeParam` structure. The returned structure represents the list of parameters required by the AllocRequantize color allocation technique and may be used as the *color_alloc_param* argument of `XieFloConvertToIndex()` (when the *color_alloc_tech* argument is `xieValColorAllocRequantize`).

If insufficient memory is available, `XieTecColorAllocRequantize()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The AllocRequantize color allocation technique first reduces the total number of discrete pixel values in the image to be no more than a specified number, and then allocates the resulting pixel values as read-only cells from the Colormap. One method of accomplishing this reduction process can be found in "Color image quantization for frame buffer display", Heckbert, P. S., *Comput. Graph.* 16, 3.

If *max_cells* is zero or greater than the number of unallocated Colormap cells, the reduction algorithm will restrict its output to the number of free cells. A `ColorAlloc` event, which warns the client that results are of lesser fidelity than desired, can be sent if the number of pixels had to be restricted to a lesser number than *max_cells* due to a lack of free Colormap cells. The AllocRequantize color allocation technique is only appropriate for dynamic Colormaps.

Structures

`XieTecColorAllocRequantize()` sets the structure field `max_cells` to the value of the argument *max_cells*.

```
typedef struct {
    unsigned long max_cells;
} XieColorAllocRequantizeParam;
```

See Also

XieFloConvertToIndex().

Name

XieTecRGBToCIELab - allocate and fill an `XieRGBToCIELabParam` structure.

Synopsis

```
XieRGBToCIELabParam *XieTecRGBToCIELab (matrix, white_adjust_tech,
            white_adjust_param)
   XieMatrix matrix;
   XieWhiteAdjustTechnique white_adjust_tech;
   XiePointer white_adjust_param;
```

Arguments

matrix	Specifies the conversion matrix.
white_adjust_tech	Specifies the WhiteAdjust technique to be used.
white_adjust_param	Specifies the list of parameters required by *white_adjust_tech*.

Returns

The `XieRGBToCIELabParam` structure.

Description

`XieTecRGBToCIELab()` allocates and returns a pointer to an `XieRGBToCIELabParam` structure. The returned structure represents the list of parameters required by the RGBToCIELab color conversion technique and may be used as the *color_param* argument of `XieFloConvertFromRGB()` (when the *color_space* argument is `xieValRGBToCIELab`).

If insufficient memory is available, `XieTecRGBToCIELab()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

`XieTecRGBToCIELab()` converts RGB data to the CIELab colorspace, an international standard designed for perceptual uniformity. The colorspace coordinates are denoted by *L*, *a*, and *b*, and is defined in: CIE. Recommendations on Uniform Color Spaces, Color-Difference Equations, Psychometric Color Terms. Bureau Central de la CIE (Supplement 2 of CIE Publication 15 (E-1.3.1) 1971) 1978.

matrix is a 3x3 RGB to CIEXYZ conversion matrix (the source white point is also encoded in *matrix*). *white_adjust_tech* is the WhiteAdjust technique that can be used to shift the white point of the output data. *white_adjust_param* is the list of parameters required by *white_adjust_tech*.

The input data type can be constrained or unconstrained; the output data type is always unconstrained. When the input is constrained, the data are normalized to the range [0, 1] (i.e., scaled by 1/(levels - 1) prior to the conversion).

WhiteAdjust techniques define the white point adjustment techniques which can be used when converting to or from the `RGB` colorspace. One of the following standard WhiteAdjust technique values can be assigned to *white_adjust_tech*:

```
xieValWhiteAdjustDefault
xieValWhiteAdjustNone
xieValWhiteAdjustCIELabShift
```

If a vendor defined additional private WhiteAdjust techniques, the private technique values given to these techniques can be assigned to *white_adjust_tech*.

The server is required to support the default technique which is bound to one of the standard techniques defined above or a private technique.

Structures

`XieTecRGBToCIELab()` sets the structure field `matrix` to the values of the argument *matrix*; the structure field `white_adjust_tech` to the value of the argument *white_adjust_tech*; and the structure field `white_adjust_param` to the value of the argument *white_adjust_param*.

```
typedef float XieMatrix[9];
typedef unsigned XieWhiteAdjustTechnique;
typedef struct {
    XieMatrix matrix;
    XieWhiteAdjustTechnique white_adjust_tech;
    XiePointer white_adjust_param;
} XieRGBToCIELabParam;

/* Definitions for WhiteAdjust Techniques */
#define xieValWhiteAdjustDefault       0
#define xieValWhiteAdjustNone          1
#define xieValWhiteAdjustCIELabShift   2
```

See Also

XieFloConvertFromRGB(), XieTecWhiteAdjustCIELabShift().

Name

XieTecRGBToCIEXYZ - allocate and fill an `XieRGBToCIEXYZParam` structure.

Synopsis

```
XieRGBToCIEXYZParam *XieTecRGBToCIEXYZ (matrix, white_adjust_tech,
            white_adjust_param)
   XieMatrix matrix;
   XieWhiteAdjustTechnique white_adjust_tech;
   XiePointer white_adjust_param;
```

Arguments

matrix	Specifies the conversion matrix.
white_adjust_tech	Specifies the WhiteAdjust technique to be used.
white_adjust_param	Specifies the list of parameters required by *white_adjust_tech*.

Returns

The `XieRGBToCIEXYZParam` structure.

Description

`XieTecRGBToCIEXYZ()` allocates and returns a pointer to an `XieRGBToCIEXYZParam` structure. The returned structure represents the list of parameters required by the RGBToCIEXYZ color conversion technique and may be used as the *color_param* argument of `XieFloConvertFromRGB()` (when the *color_space* argument is `xieValRGBToCIEXYZ`).

If insufficient memory is available, `XieTecRGBToCIEXYZ()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

`XieTecRGBToCIEXYZ()` converts RGB data to the CIEXYZ colorspace, an international standard designed for perceptual uniformity, which is defined in: CIE. Recommendations on Uniform Color Spaces, Color-Difference Equations, Psychometric Color Terms. Bureau Central de la CIE (Supplement 2 of CIE Publication 15 (E-1.3.1) 1971) 1978.

matrix is a 3x3 RGB to CIEXYZ conversion matrix (the source white point is also encoded in *matrix*). *white_adjust_tech* is the WhiteAdjust technique that can be used to shift the white point of the output data. *white_adjust_param* is the list of parameters required by *white_adjust_tech*.

The input data type can be constrained or unconstrained; the output data type is always unconstrained. When the input is constrained, the data are normalized to the range [0, 1] (i.e., scaled by 1/(levels - 1) prior to the conversion).

WhiteAdjust techniques define the white point adjustment techniques which can be used when converting to or from the `RGB` colorspace. One of the following standard WhiteAdjust technique values can be assigned to *white_adjust_tech*:

```
xieValWhiteAdjustDefault
xieValWhiteAdjustNone
xieValWhiteAdjustCIELabShift
```

If a vendor defined additional private WhiteAdjust techniques, the private technique values given to these techniques can be assigned to *white_adjust_tech*.

The server is required to support the default technique which is bound to one of the standard techniques defined above or a private technique.

Structures

`XieTecRGBToCIEXYZ()` sets the structure field `matrix` to the values of the argument *matrix*; the structure field `white_adjust_tech` to the value of the argument *white_adjust_tech*; and the structure field `white_adjust_param` to the value of the argument *white_adjust_param*.

```
typedef float XieMatrix[9];
typedef unsigned XieWhiteAdjustTechnique;
typedef struct {
    XieMatrix matrix;
    XieWhiteAdjustTechnique white_adjust_tech;
    XiePointer white_adjust_param;
} XieRGBToCIEXYZParam;

/* Definitions for WhiteAdjust Techniques */
#define xieValWhiteAdjustDefault       0
#define xieValWhiteAdjustNone          1
#define xieValWhiteAdjustCIELabShift   2
```

See Also

XieFloConvertFromRGB(), XieTecWhiteAdjustCIELabShift().

Name

XieTecRGBToYCbCr - allocate and fill an `XieRGBToYCbCrParam` structure.

Synopsis

```
XieRGBToYCbCrParam *XieTecRGBToYCbCr (levels, luma_red, luma_green,
            luma_blue, bias)
   XieLevels levels;
   double luma_red;
   double luma_green;
   double luma_blue;
   XieConstant bias;
```

Arguments

levels	Specifies the output levels.
luma_red	Specifies the proportion of red in the luminance band.
luma_green	Specifies the proportion of green in the luminance band.
luma_blue	Specifies the proportion of blue in the luminance band.
bias	Specifies an offset to add to the output pixels values.

Returns

The `XieRGBToYCbCrParam` structure.

Description

`XieTecRGBToYCbCr()` allocates and returns a pointer to an `XieRGBToYCbCrParam` structure. The returned structure represents the list of parameters required by the RGBToYCbCr color conversion technique and may be used as the *color_param* argument of `XieFloConvertFromRGB()` (when the *color_space* argument is `xieValRGBToYCbCr`).

If insufficient memory is available, `XieTecRGBToYCbCr()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

`XieTecRGBToYCbCr()` converts RGB data to the YCbCr colorspace. Source data may be constrained or unconstrained; the output type will match. If the source data is constrained, *levels* determines the output levels; otherwise *levels* is ignored.

Structures

`XieTecRGBToYCbCr()` sets the structure field `levels` to the values of the argument *levels*; the structure fields `luma_red`, `luma_green`, `luma_blue` to the values of the arguments *luma_red*, *luma_green*, *luma_blue*; and the structure field `bias` to the values of the argument *bias*.

```
typedef float XieConstant[3];
typedef unsigned long XieLevels[3];
typedef struct {
   XieLevels levels;
   float luma_red;
   float luma_green;
   float luma_blue;
   XieConstant bias;
```

```
} XieRGBToYCbCrParam;
```

See Also

XieFloConvertFromRGB().

Name

XieTecRGBToYCC - allocate and fill an `XieRGBToYCCParam` structure.

Synopsis

```
XieRGBToYCCParam *XieTecRGBToYCC (levels, luma_red, luma_green,
            luma_blue, scale)
   XieLevels levels;
   double luma_red;
   double luma_green;
   double luma_blue;
   double scale;
```

Arguments

levels	Specifies the output levels.
luma_red	Specifies the proportion of red in the luminance band.
luma_green	Specifies the proportion of green in the luminance band.
luma_blue	Specifies the proportion of blue in the luminance band.
scale	Specifies a compression factor to apply to the output pixels values.

Returns

The `XieRGBToYCCParam` structure.

Description

`XieTecRGBToYCC()` allocates and returns a pointer to an `XieRGBToYCCParam` structure. The returned structure represents the list of parameters required by the RGBToYCC color conversion technique and may be used as the *color_param* argument of `XieFloConvertFromRGB()` (when the *color_space* argument is `xieValRGBToYCC`).

If insufficient memory is available, `XieTecRGBToYCC()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

`XieTecRGBToYCC()` converts RGB data to the YCC colorspace. The PhotoYCC color-encoding scheme is defined in: KODAK PhotoCD System - A Planning Guide for Developers, Eastman Kodak Co., Part no. DCI200R, 1991.

Source data may be constrained or unconstrained; the output type will match. If the source data is constrained, *levels* determines the output levels; otherwise *levels* is ignored. Typical values cited in the literature for *scale* are in the range of about 1.35 to 1.4.

Structures

`XieTecRGBToYCC()` sets the structure field `levels` to the values of the argument *levels*; the structure fields `luma_red`, `luma_green`, `luma_blue` are set to the values of the arguments *luma_red, luma_green, luma_blue*; and the structure field `scale` to the value of the argument *scale*.

```
typedef unsigned long XieLevels[3];
typedef struct {
   XieLevels levels;
   float luma_red;
```

```
    float luma_green;
    float luma_blue;
    float scale;
} XieRGBToYCCParam;
```

See Also

XieFloConvertFromRGB().

Name

XieTecCIELabToRGB - allocate and fill an `XieCIELabToRGBParam` structure.

Synopsis

```
XieCIELabToRGBParam *XieTecCIELabToRGB (matrix, white_adjust_tech,
            white_adjust_param, gamut_tech, gamut_param)
   XieMatrix matrix;
   XieWhiteAdjustTechnique white_adjust_tech;
   XiePointer white_adjust_param;
   XieGamutTechnique gamut_tech;
   XiePointer gamut_param;
```

Arguments

matrix	Specifies the conversion matrix.
white_adjust_tech	Specifies the WhiteAdjust technique to be used.
white_adjust_param	Specifies the list of parameters required by *white_adjust_tech*.
gamut_tech	Specifies the Gamut technique to be used.
gamut_param	Specifies the list of parameters required by *gamut_tech*.

Returns

The `XieCIELabToRGBParam` structure.

Description

`XieTecCIELabToRGB()` allocates and returns a pointer to an `XieCIELabToRGBParam` structure. The returned structure represents the list of parameters required by the CIELabToRGB color conversion technique and may be used as the *color_param* argument of `XieFloConvertToRGB()` (when the *color_space* argument is `xieValCIELabToRGB`).

If insufficient memory is available, `XieTecCIELabToRGB()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

`XieTecCIELabToRGB()` converts CIELab data to the RGB colorspace. The CIELab colorspace is an international standard designed for perceptual uniformity. The colorspace coordinates are denoted by *L*, *a*, and *b*, and is defined in: CIE. Recommendations on Uniform Color Spaces, Color-Difference Equations, Psychometric Color Terms. Bureau Central de la CIE (Supplement 2 of CIE Publication 15 (E-1.3.1) 1971) 1978.

matrix is a 3x3 CIEXYZ to RGB conversion matrix (the target white point is also encoded in *matrix*). *white_adjust_tech* is the WhiteAdjust technique that can be used to shift the white point of the source data prior to conversion. *white_adjust_param* is the list of parameters required by *white_adjust_tech*. *gamut_tech* is the Gamut technique that can be used to keep the output pixels within the bounds of the RGB colorspace. *gamut_param* is the list of parameters required by *gamut_tech*.

The input data type must be unconstrained; the output data type is also unconstrained.

WhiteAdjust techniques define the white point adjustment techniques which can be used when converting to or from the `RGB` colorspace. One of the following standard WhiteAdjust technique values can be assigned to *white_adjust_tech*:

```
xieValWhiteAdjustDefault
xieValWhiteAdjustNone
xieValWhiteAdjustCIELabShift
```

If a vendor defined additional private WhiteAdjust techniques, the private technique values given to these techniques can be assigned to *white_adjust_tech*.

The server is required to support the default technique which is bound to one of the standard techniques defined above or a private technique.

Gamut techniques define the gamut compression techniques used to deal with converted colors which lie outside the gamut of the `RGB` space. One of the following standard gamut technique values can be assigned to *gamut_tech*:

```
xieValGamutDefault
xieValGamutNone
xieValGamutClipRGB
```

If a vendor defined additional private gamut techniques, the private technique values given to these techniques can be assigned to *gamut_tech*.

The server is required to support the default technique which is bound to one of the standard techniques defined above or a private technique.

Structures

`XieTecCIELabToRGB()` sets the structure field `matrix` to the values of the argument *matrix*; the structure field `white_adjust_tech` to the value of the argument *white_adjust_tech*; the structure field `white_adjust_param` to the value of the argument *white_adjust_param*; the structure field `gamut_tech` to the value of the argument *gamut_tech*; and the structure field `gamut_param` to the value of the argument *gamut_param*.

```
typedef float XieMatrix[9];
typedef unsigned XieGamutTechnique;
typedef unsigned XieWhiteAdjustTechnique;
typedef struct {
   XieMatrix matrix;
   XieWhiteAdjustTechnique white_adjust_tech;
   XiePointer white_adjust_param;
   XieGamutTechnique gamut_tech;
   XiePointer gamut_param;
} XieCIELabToRGBParam;

/* Definitions for WhiteAdjust Techniques */
#define xieValWhiteAdjustDefault       0
#define xieValWhiteAdjustNone          1
#define xieValWhiteAdjustCIELabShift   2

/* Definitions for Gamut Techniques */
#define xieValGamutDefault             0
#define xieValGamutNone                1
#define xieValGamutClipRGB             2
```

See Also

XieFloConvertToRGB(), XieTecWhiteAdjustCIELabShift().

Name

XieTecCIEXYZToRGB - allocate and fill an `XieCIEXYZToRGBParam` structure.

Synopsis

```
XieCIEXYZToRGBParam *XieTecCIEXYZToRGB (matrix, white_adjust_tech,
            white_adjust_param, gamut_tech, gamut_param)
   XieMatrix matrix;
   XieWhiteAdjustTechnique white_adjust_tech;
   XiePointer white_adjust_param;
   XieGamutTechnique gamut_tech;
   XiePointer gamut_param;
```

Arguments

matrix	Specifies the conversion matrix.
white_adjust_tech	Specifies the WhiteAdjust technique to be used.
white_adjust_param	Specifies the list of parameters required by *white_adjust_tech*.
gamut_tech	Specifies the Gamut technique to be used.
gamut_param	Specifies the list of parameters required by *gamut_tech*.

Returns

The `XieCIEXYZToRGBParam` structure.

Description

`XieTecCIEXYZToRGB()` allocates and returns a pointer to an `XieCIEXYZToRGBParam` structure. The returned structure represents the list of parameters required by the CIEXYZToRGB color conversion technique and may be used as the *color_param* argument of `XieFloConvertToRGB()` (when the *color_space* argument is `xieValCIEXYZToRGB`).

If insufficient memory is available, `XieTecCIEXYZToRGB()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

`XieTecCIEXYZToRGB()` converts CIEXYZ data to the RGB colorspace. The CIEXYZ colorspace is an international standard designed for perceptual uniformity, which is defined in: CIE. Recommendations on Uniform Color Spaces, Color-Difference Equations, Psychometric Color Terms. Bureau Central de la CIE (Supplement 2 of CIE Publication 15 (E-1.3.1) 1971) 1978.

matrix is a 3x3 CIEXYZ to RGB conversion matrix (the target white point is also encoded in *matrix*). *white_adjust_tech* is the WhiteAdjust technique that can be used to shift the white point of the source data prior to conversion. *white_adjust_param* is the list of parameters required by *white_adjust_tech*. *gamut_tech* is the Gamut technique that can be used to keep the output pixels within the bounds of the RGB colorspace. *gamut_param* is the list of parameters required by *gamut_tech*.

The input data type must be unconstrained; the output data type is also unconstrained.

WhiteAdjust techniques define the white point adjustment techniques which can be used when converting to or from the `RGB` colorspace. One of the following standard WhiteAdjust technique values can be assigned to *white_adjust_tech*:

```
xieValWhiteAdjustDefault
xieValWhiteAdjustNone
xieValWhiteAdjustCIELabShift
```

If a vendor defined additional private WhiteAdjust techniques, the private technique values given to these techniques can be assigned to *white_adjust_tech*.

The server is required to support the default technique which is bound to one of the standard techniques defined above or a private technique.

Gamut techniques define the gamut compression techniques used to deal with converted colors which lie outside the gamut of the RGB space. One of the following standard gamut technique values can be assigned to *gamut_tech*:

```
xieValGamutDefault
xieValGamutNone
xieValGamutClipRGB
```

If a vendor defined additional private gamut techniques, the private technique values given to these techniques can be assigned to *gamut_tech*.

The server is required to support the default technique which is bound to one of the standard techniques defined above or a private technique.

Structures

`XieTecCIEXYZToRGB()` sets the structure field `matrix` to the values of the argument *matrix*; the structure field `white_adjust_tech` to the value of the argument *white_adjust_tech*; the structure field `white_adjust_param` to the value of the argument *white_adjust_param*; the structure field `gamut_tech` to the value of the argument *gamut_tech*; and the structure field `gamut_param` to the value of the argument *gamut_param*.

```
typedef float XieMatrix[9];
typedef unsigned XieGamutTechnique;
typedef unsigned XieWhiteAdjustTechnique;
typedef struct {
   XieMatrix matrix;
   XieWhiteAdjustTechnique white_adjust_tech;
   XiePointer white_adjust_param;
   XieGamutTechnique gamut_tech;
   XiePointer gamut_param;
} XieCIEXYZToRGBParam;

/* Definitions for WhiteAdjust Techniques */
#define xieValWhiteAdjustDefault       0
#define xieValWhiteAdjustNone          1
#define xieValWhiteAdjustCIELabShift   2

/* Definitions for Gamut Techniques */
#define xieValGamutDefault             0
#define xieValGamutNone                1
#define xieValGamutClipRGB             2
```

See Also

XieFloConvertToRGB(), XieTecWhiteAdjustCIELabShift().

Name

XieTecYCbCrToRGB - allocate and fill an `XieYCbCrToRGBParam` structure.

Synopsis

```
XieYCbCrToRGBParam *XieTecYCbCrToRGB (levels, luma_red, luma_green,
            luma_blue, bias, gamut_tech, gamut_param)
   XieLevels levels;
   double luma_red;
   double luma_green;
   double luma_blue;
   XieConstant bias;
   XieGamutTechnique gamut_tech;
   XiePointer gamut_param;
```

Arguments

levels	Specifies the output levels.
luma_red	Specifies the proportion of red in the luminance band (Y).
luma_green	Specifies the proportion of green in the luminance band (Y).
luma_blue	Specifies the proportion of blue in the luminance band (Y).
bias	Specifies an offset to remove from the source pixels values.
gamut_tech	Specifies the Gamut technique to be used.
gamut_param	Specifies the list of parameters required by *gamut_tech*.

Returns

The `XieYCbCrToRGBParam` structure.

Description

`XieTecYCbCrToRGB()` allocates and returns a pointer to an `XieYCbCrToRGBParam` structure. The returned structure represents the list of parameters required by the YCbCrToRGB color conversion technique and may be used as the *color_param* argument of `XieFloConvertToRGB()` (when the *color_space* argument is `xieValYCbCrToRGB`).

If insufficient memory is available, `XieTecYCbCrToRGB()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

`XieTecYCbCrToRGB()` converts YCbCr data to the RGB colorspace. Source data may be constrained or unconstrained; the output type will match. If the source data is constrained, *levels* determines the output levels; otherwise *levels* is ignored.

Gamut techniques define the gamut compression techniques used to deal with converted colors which lie outside the gamut of the `RGB` space. One of the following standard gamut technique values can be assigned to *gamut_tech*:

```
xieValGamutDefault
xieValGamutNone
xieValGamutClipRGB
```

If a vendor defined additional private gamut techniques, the private technique values given to these techniques can be assigned to *gamut_tech*.

The server is required to support the default technique which is bound to one of the standard techniques defined above, or a private technique.

Structures

`XieTecYCbCrToRGB()` sets the structure field `levels` to the values of the argument *`levels`*; the structure fields `luma_red`, `luma_green`, `luma_blue` to the values of the arguments *`luma_red`*, *`luma_green`*, *`luma_blue`*; the structure field `bias` to the values of the argument *`bias`*; the structure field `gamut_tech` to the value of the argument *`gamut_tech`*; and the structure field `gamut_param` to the value of the argument *`gamut_param`*.

```
typedef float XieConstant[3];
typedef unsigned long XieLevels[3];
typedef unsigned XieGamutTechnique;
typedef struct {
   XieLevels levels;
   float luma_red;
   float luma_green;
   float luma_blue;
   XieConstant bias;
   XieGamutTechnique gamut_tech;
   XiePointer gamut_param;
} XieYCbCrToRGBParam;

/* Definitions for Gamut Techniques */
#define xieValGamutDefault            0
#define xieValGamutNone               1
#define xieValGamutClipRGB            2
```

See Also

XieFloConvertToRGB().

Name

XieTecYCCToRGB - allocate and fill an `XieYCCToRGBParam` structure.

Synopsis

```
XieYCCToRGBParam *XieTecYCCToRGB (levels, luma_red, luma_green,
            luma_blue, scale, gamut_tech, gamut_param)
   XieLevels levels;
   double luma_red;
   double luma_green;
   double luma_blue;
   double scale;
   XieGamutTechnique gamut_tech;
   XiePointer gamut_param;
```

Arguments

levels	Specifies the output levels.
luma_red	Specifies the proportion of red in the luminance band (Y).
luma_green	Specifies the proportion of green in the luminance band (Y).
luma_blue	Specifies the proportion of blue in the luminance band (Y).
scale	Specifies an expansion factor to apply to the output pixels values.
gamut_tech	Specifies the Gamut technique to be used.
gamut_param	Specifies the list of parameters required by *gamut_tech*.

Returns

The `XieYCCToRGBParam` structure.

Description

`XieTecYCCToRGB()` allocates and returns a pointer to an `XieYCCToRGBParam` structure. The returned structure represents the list of parameters required by the YCCToRGB color conversion technique and may be used as the *color_param* argument of `XieFloConvertToRGB()` (when the *color_space* argument is `xieValYCCToRGB`).

If insufficient memory is available, `XieTecYCCToRGB()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

`XieTecYCCToRGB()` converts YCC data to the RGB colorspace. The PhotoYCC color-encoding scheme is defined in: KODAK PhotoCD System - A Planning Guide for Developers, Eastman Kodak Co., Part no. DCI200R, 1991.

Source data may be constrained or unconstrained; the output type will match. If the source data is constrained, *levels* determines the output levels; otherwise *levels* is ignored. Typical values cited in the literature for *scale* are in the range of about 1.35 to 1.4.

Gamut techniques define the gamut compression techniques used to deal with converted colors which lie outside the gamut of the `RGB` space. One of the following standard gamut technique values can be assigned to *gamut_tech*:

```
xieValGamutDefault
xieValGamutNone
xieValGamutClipRGB
```

If a vendor defined additional private gamut techniques, the private technique values given to these techniques can be assigned to *gamut_tech* .

The server is required to support the default technique which is bound to one of the standard techniques defined above or a private technique.

Structures

`XieTecYCCToRGB()` sets the structure field `levels` to the values of the argument *levels*; the structure fields `luma_red`, `luma_green`, `luma_blue` to the values of the arguments *luma_red, luma_green, luma_blue*; the structure field `scale` to the values of the argument *scale*; the structure field `gamut_tech` to the value of the argument *gamut_tech*; and the structure field `gamut_param` to the value of the argument *gamut_param*.

```
typedef unsigned long XieLevels[3];
typedef unsigned XieGamutTechnique;
typedef struct {
   XieLevels levels;
   float luma_red;
   float luma_green;
   float luma_blue;
   float scale;
   XieGamutTechnique gamut_tech;
   XiePointer gamut_param;
} XieYCCToRGBParam;

/* Definitions for Gamut Techniques */
#define xieValGamutDefault            0
#define xieValGamutNone               1
#define xieValGamutClipRGB            2
```

See Also

XieFloConvertToRGB().

Name

XieTecClipScale - allocate and fill an `XieClipScaleParam` structure.

Synopsis

```
XieClipScaleParam *XieTecClipScale (in_low, in_high, out_low,
            out_high)
   XieConstant in_low;
   XieConstant in_high;
   XieLTriplet out_low;
   XieLTriplet out_high;
```

Arguments

in_low	Specifies an input pixel limit.
in_high	Specifies an input pixel limit.
out_low	Specifies an output pixel limit.
out_high	Specifies an output pixel limit.

Returns

The `XieClipScaleParam` structure.

Description

`XieTecClipScale()` allocates and returns a pointer to an `XieClipScaleParam` structure. The returned structure represents the list of parameters required by the constrain technique and may be used as the *constrain_param* argument of `XieFloConstrain()` (when the *constrain_tech* argument is `xieValClipScale`).

If insufficient memory is available, `XieTecClipScale()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

For each band, output pixels will be clipped to the range [*out_low*, *out_high*]. If *in_low* is less than *in_high*, then all pixels less than or equal to *in_low* will map to *out_low*, and all pixels that are greater than or equal to *in_high* will map to *out_high*. All intermediate pixel values are scaled proportionately to the output range. Non-integral output values are rounded to the nearest integer.

If *in_low* is greater than *in_high* then all pixels that are greater than or equal to *in_low* will map to *out_low*, and all pixels that are less than or equal to *in_high* will map to *out_high*. All intermediate pixel values will be linearly mapped to the output range, such that *in_low* maps to *out_low*, and *in_high* maps to *out_high*. Non-integral output values are rounded to the nearest integer.

in_low should not equal *in_high*, *out_low* should be less than *out_high*, and *out_high* should not exceed levels-1.

Structures

`XieTecClipScale()` sets the structure fields `input_low`, `input_high` to the values of the arguments *in_low, in_high*; and the structure fields `output_low`, `output_high` to the values of the arguments *out_low, out_high*.

```
typedef float XieConstant[3];
typedef unsigned long XieLTriplet[3];
typedef struct {
   XieConstant input_low,input_high;
   XieLTriplet output_low,output_high;
} XieClipScaleParam;
```

See Also

XieFloConstrain()

Name

XieTecConvolveConstant - allocate and fill an `XieConvolveConstantParam` structure.

Synopsis

```
XieConvolveConstantParam *XieTecConvolveConstant (constant)
   XieConstant constant;
```

Arguments

constant	Specifies the value to use if pixels are required from beyond the edge of the image.

Returns

The `XieConvolveConstantParam` structure.

Description

`XieTecConvolveConstant()` allocates and returns a pointer to an `XieConvolveConstantParam` structure. The returned structure represents the list of parameters required by the convolve technique and may be used as the *convolve_param* argument of `XieFloConvolve()` (when the *convolve_tech* argument is `xieValConvolveConstant`).

If insufficient memory is available, `XieTecConvolveConstant()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

Various methods of handling edge conditions are provided for convolve techniques. These techniques determine what pixel values are used when the convolve technique requires data beyond the image bounds. Convolve techniques only come into play when the kernel is positioned partially off the edge of the image. Data around the edges of a *process domain* are convolved with adjacent image pixels wherever possible. A *process domain* is inserted in many element definitions and is used to restrict the element's processing to a subset of the source data pixels; it can be either a list-of-rectangles or a control-plane.

The Constant Convolve technique uses the value specified by *constant* if pixels are required from beyond the edge of the image.

Structures

`XieTecConvolveConstant()` sets the structure field `constant` to the value of the argument *constant*.

```
typedef float XieConstant[3];
typedef struct {
   XieConstant constant;
} XieConvolveConstantParam;
```

See Also

XieFloConvolve().

Name

XieTecDecodeUncompressedSingle - allocate and fill an `XieDecodeUncompressedSingleParam` structure.

Synopsis

```
XieDecodeUncompressedSingleParam *XieTecDecodeUncompressedSingle
            (fill_order, pixel_order, pixel_stride, left_pad,
            scanline_pad)
   XieOrientation fill_order;
   XieOrientation pixel_order;
   unsigned int pixel_stride;
   unsigned int left_pad;
   unsigned int scanline_pad;
```

Arguments

fill_order	Specifies the method of pixel packing.
pixel_order	Specifies pixel ordering within the data stream.
pixel_stride	Specifies the number of bits between consecutive pixels within a scanline.
left_pad	Specifies the number of pad bits in each scanline.
scanline_pad	Specifies a multiple of bytes to which each scanline is padded.

Returns

The `XieDecodeUncompressedSingleParam` structure.

Description

`XieDecodeUncompressedSingle()` allocates and returns a pointer to an `XieDecodeUncompressedSingleParam` structure. The returned structure represents the list of parameters required by the decode technique and may be used as the *decode_param* argument of `XieFloImportClientPhoto()` (when the *decode_tech* argument is `xieValDecodeUncompressedSingle`).

If insufficient memory is available, `XieTecDecodeUncompressedSingle()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The decode uncompressed single technique is used when no compression scheme has been applied to single band image data. The parameters define the format of the data stream of uncompressed data (the server may reformat the data as it chooses prior to processing or storage). When multiple pixels are put in the same byte, or a pixel spans multiple bytes, *fill_order* specifies whether the pixels (or parts of pixels) are packed into the most or least significant bits of a byte first. For pixels that span a byte boundary, *pixel_order* defines whether the most or least significant bits of the pixel are transported first within the data stream. One of the following standard orientation values can be assigned to *fill_order* and *pixel_order*:

```
xieValLSFirst
xieValMSFirst
```

The table below shows the relationship between *fill_order* and *pixel_order*, using two 10-bit pixels, each with two bits of pad (within each pixel the LS-bits are “0” and “a”, the MS-bits are “9” and “j”, the pad bits are “p”).

fill-order	LSFirst (pixel-order)	MSFirst (pixel-order)
LSFirst	76543210 dcbapp98 ppjihgfe	98765432 jihgpp10 ppfedcba
MSFirst	76543210 98ppdcba jihgfepp	98765432 10ppjihg fedcbapp

pixel_stride is the number of bits between the start of consecutive pixels within a scanline; it must be at least enough bits to contain the number of input levels. *left_pad* is the number of pad bits preceding the first image pixel in each scanline; if the server's Alignment attribute is Alignable, or *pixel_stride* fits the definition of Alignable, the value of *left_pad* must be a multiple of *pixel_stride* or a multiple of 8; otherwise, *left_pad* may be any arbitrary value. *scanline_pad* defines a multiple of bytes to which each scanline is padded; valid values are: 0 (not aligned), 1, 2, 4, 8, and 16. The total number of bits-per-scanline in the data stream includes: *left_pad*, the image data (*width* x *pixel_stride*), and sufficient additional bits to satisfy *scanline_pad*.

Structures

XieDecodeUncompressedSingle() sets the structure field fill_order to the value of the argument *fill_order*; the structure field pixel_order to the value of the argument *pixel_order*; the structure field pixel_stride to the value of the argument *pixel_stride*; the structure field left_pad to the value of the argument *left_pad*; and the structure field scanline_pad to the value of the argument *scanline_pad*.

```
typedef unsigned XieOrientation;
typedef struct {
   XieOrientation fill_order;
   XieOrientation pixel_order;
   unsigned int pixel_stride;
   unsigned int left_pad;
   unsigned int scanline_pad;
} XieDecodeUncompressedSingleParam;

/* Definitions of Orientation Types */
#define xieValLSFirst                    1
#define xieValMSFirst                    2
```

See Also

XieFloImportClientPhoto(), XiePutClientData().

Name

XieTecDecodeUncompressedTriple - allocate and fill an `XieDecodeUncompressedTripleParam` structure.

Synopsis

```
XieDecodeUncompressedTripleParam *XieTecDecodeUncompressedTriple
            (fill_order, pixel_order, band_order, interleave,
            pixel_stride, left_pad, scanline_pad)
   XieOrientation fill_order;
   XieOrientation pixel_order;
   XieOrientation band_order;
   XieInterleave interleave;
   unsigned char pixel_stride[3];
   unsigned char left_pad[3];
   unsigned char scanline_pad[3];
```

Arguments

fill_order	Specifies the method of pixel packing.
pixel_order	Specifies pixel ordering within the data stream.
band_order	Specifies the order of the image data sent through the protocol stream.
interleave	Specifies how the image bands are interleaved.
pixel_stride	Specifies the number of bits between consecutive pixels within a scanline.
left_pad	Specifies the number of pad bits in each scanline.
scanline_pad	Specifies a multiple of bytes to which each scanline is padded.

Returns

The `XieDecodeUncompressedTripleParam` structure.

Description

`XieDecodeUncompressedTriple()` allocates and returns a pointer to an `XieDecodeUncompressedTripleParam` structure. The returned structure represents the list of parameters required by the decode technique and may be used as the *decode_param* argument of `XieFloImportClientPhoto()` (when the *decode_tech* argument is `xieValDecodeUncompressedTriple`).

If insufficient memory is available, `XieTecDecodeUncompressedTriple()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The decode uncompressed triple technique is used when no compression scheme has been applied to triple band image data. The parameters define the format of the data stream of uncompressed data (the server may reformat the data as it chooses prior to processing or storage). When multiple pixels are put in the same byte, or a pixel spans multiple bytes, *fill_order* specifies whether the pixels (or parts of pixels) are packed into the most or least significant bits of a byte first. For pixels that span a byte boundary, *pixel_order* defines whether the most or least significant bits of the pixel are transported first within the data stream.

One of the following standard orientation values can be assigned to *fill_order* and *pixel_order*:

```
xieValLSFirst
xieValMSFirst
```

The table below shows the relationship between *fill_order and pixel_order*, using two 10-bit pixels, each with two bits of pad (within each pixel the LS-bits are “0” and “a”, the MS-bits are “9” and “j”, the pad bits are “p”)

fill-order	LSFirst (pixel-order)	MSFirst (pixel-order)
LSFirst	76543210 dcbapp98 ppjihgfe	98765432 jihgpp10 ppfedcba
MSFirst	76543210 98ppdcba jihgfepp	98765432 10ppjihg fedcbapp

pixel_stride is the number of bits between the start of consecutive pixels within a scanline; It must be at least enough bits to contain the number of input levels. *left_pad* is the number of pad bits preceding the first image pixel in each scanline; if the server's Alignment attribute is Alignable, or *pixel_stride* fits the definition of Alignable, the value of *left_pad* must be a multiple of *pixel_stride* or a multiple of 8; otherwise, *left_pad* may be any arbitrary value. *scanline_pad* defines a multiple of bytes to which each scanline is padded; valid values are: 0 (not aligned), 1, 2, 4, 8, and 16. The total number of bits-per-scanline in the data stream includes: *left_pad*, the image data (*width* x *pixel_stride*), and sufficient additional bits to satisfy *scanline_pad*.

interleave describes how the image bands are interleaved (per pixel within a single plane, or sent as three separate planes); if *interleave* is `xieValBandByPixel`, inter-band dimensions must match: the widths and the heights of all bands must match. One of the following standard interleave values can be assigned to *interleave*:

```
xieValBandByPixel
xieValBandByPlane
```

band_order is the order of the image bands or image planes as they are transmitted through the protocol stream. *band_order* can be set to one of the standard orientation values. The least significant band of trichromatic data is the first band mentioned in the common name of the colorspace : red is the least significant band of `RGB` data. For band-by-pixel data, *band_order* specifies whether this band is put in the least or most significant bits of a pixel:

LSFirst	MSFirst
$B_1B_0G_2G_1G_0R_2R_1R_0$	$R_2R_1R_0G_2G_1G_0B_1B_0$

For band-by-plane data, *band_order* specifies whether this band corresponds with the least significant or most significant image plane. Each plane is transported as a separate data stream:

band	LSFirst	MSFirst
0	$R_7R_6R_5R_4R_3R_2R_1R_0$	$B_7B_6B_5B_4B_3B_2B_1B_0$
1	$G_7G_6G_5G_4G_3G_2G_1G_0$	$G_7G_6G_5G_4G_3G_2G_1G_0$
2	$B_7B_6B_5B_4B_3B_2B_1B_0$	$R_7R_6R_5R_4R_3R_2R_1R_0$

Structures

`XieDecodeUncompressedTriple()` sets the structure field `fill_order` to the value of the argument *`fill_order`*; the structure field `pixel_order` to the value of the argument *`pixel_order`*; the structure field `band_order` to the value of the argument *`band_order`*; the structure field `interleave` to the value of the argument *`interleave`*; the structure field `pixel_stride` to the value of the argument *`pixel_stride`*; the structure field `left_pad` to the value of the argument *`left_pad`*; and the structure field `scanline_pad` to the value of the argument *`scanline_pad`*.

```
typedef unsigned XieOrientation;
typedef unsigned XieInterleave;
typedef struct {
   unsigned char left_pad[3];
   XieOrientation fill_order;
   unsigned char pixel_stride[3];
   XieOrientation pixel_order;
   unsigned char scanline_pad[3];
   XieOrientation band_order;
   XieInterleave interleave;
} XieDecodeUncompressedTripleParam;

/* Definitions of Orientation Types */
#define xieValLSFirst                  1
#define xieValMSFirst                  2

/* Definitions for Interleave */
#define xieValBandByPixel              1
#define xieValBandByPlane              2
```

See Also

XieFloImportClientPhoto(), XiePutClientData().

Name

XieTecDecodeG31D - allocate and fill an `XieDecodeG31DParam` structure.

Synopsis

```
XieDecodeG31DParam *XieTecDecodeG31D (encoded_order, normal,
            radiometric)
   XieOrientation encoded_order;
   Bool normal;
   Bool radiometric;
```

Arguments

encoded_order	Specifies the bit-order of the encoded data.
normal	Specifies how the data was processed when it was originally encoded.
radiometric	Specifies how “white runs” are decoded.

Returns

The `XieDecodeG31DParam` structure.

Description

`XieDecodeG31D()` allocates and returns a pointer to an `XieDecodeG31DParam` structure. The returned structure represents the list of parameters required by the decode technique and may be used as the *decode_param* argument of `XieFloImportClientPhoto()` (when the *decode_tech* argument is `xieValDecodeG31D`).

If insufficient memory is available, `XieTecDecodeG31D()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

CCITT-G31D is the CCITT group 3 one dimensional encoding technique as defined by CCITT T.4, "Standardization of Group 3 Facsimile Apparatus for Document Transmission".

encoded_order specifies the bit-order of the encoded data. One of the following standard orientation values can be assigned to *encoded_order*:

```
xieValLSFirst
xieValMSFirst
```

The following table shows the encoded bit order of two bytes (within each byte the first encoded bit is “0”, last is bit “7”):

LSFirst	**MSFirst**
76543210 76543210	01234567 01234567

radiometric specifies that “white runs” in the encoded data should be represented as image ones upon decode (maximum intensity), or conversely, they will be decoded as image zeros if *radiometric* is `False`. *normal* specifies that the data was processed according to its normal fill-order when it was originally encoded.

Structures

`XieDecodeG31D()` sets the structure field `encoded_order` to the value of the argument *`encoded_order`*; the structure field `normal` to the value of the argument *`normal`*; and the structure field `radiometric` to the value of the argument *`radiometric`*.

```
typedef unsigned XieOrientation;
typedef struct {
   XieOrientation encoded_order;
   Bool normal;
   Bool radiometric;
} XieDecodeG31DParam;

/* Definitions of Orientation Types */
#define xieValLSFirst                  1
#define xieValMSFirst                  2
```

See Also

XieFloImportClientPhoto(), XiePutClientData().

Name

XieTecDecodeG32D - allocate and fill an `XieDecodeG32DParam` structure.

Synopsis

```
XieDecodeG32DParam *XieTecDecodeG32D (encoded_order, normal,
            radiometric)
   XieOrientation encoded_order;
   Bool normal;
   Bool radiometric;
```

Arguments

encoded_order	Specifies the bit-order of the encoded data.
normal	Specifies how the data was processed when it was originally encoded.
radiometric	Specifies how “white runs” are decoded.

Returns

The `XieDecodeG32DParam` structure.

Description

`XieDecodeG32D()` allocates and returns a pointer to an `XieDecodeG32DParam` structure. The returned structure represents the list of parameters required by the decode technique and may be used as the *decode_param* argument of `XieFloImportClientPhoto()` (when the *decode_tech* argument is `xieValDecodeG32D`).

If insufficient memory is available, `XieTecDecodeG32D()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

CCITT-G32D is the CCITT group 3 one dimensional encoding technique as defined by CCITT T.4, "Standardization of Group 3 Facsimile Apparatus for Document Transmission".

encoded_order specifies the bit-order of the encoded data. One of the following standard orientation values can be assigned to *encoded_order*:

```
xieValLSFirst
xieValMSFirst
```

The following table shows the encoded bit order of two bytes (within each byte the first encoded bit is “0”, last is bit “7”):

LSFirst	**MSFirst**
76543210 76543210	01234567 01234567

radiometric specifies that “white runs” in the encoded data should be represented as image ones upon decode (maximum intensity), or conversely, they will be decoded as image zeros if *radiometric* is `False`. *normal* specifies that the data was processed according to its normal fill-order when it was originally encoded.

Structures

`XieDecodeG32D()` sets the structure field `encoded_order` to the value of the argument *`encoded_order`*; the structure field `normal` to the value of the argument *`normal`*; and the structure field `radiometric` to the value of the argument *`radiometric`*.

```
typedef unsigned XieOrientation;
typedef struct {
   XieOrientation encoded_order;
   Bool normal;
   Bool radiometric;
} XieDecodeG32DParam;

/* Definitions of Orientation Types */
#define xieValLSFirst                    1
#define xieValMSFirst                    2
```

See Also

XieFloImportClientPhoto(), XiePutClientData().

Name

XieTecDecodeG42D - allocate and fill an `XieDecodeG42DParam` structure.

Synopsis

```
XieDecodeG42DParam *XieTecDecodeG42D (encoded_order, normal,
            radiometric))
   XieOrientation encoded_order;
   Bool normal;
   Bool radiometric;
```

Arguments

encoded_order	Specifies the bit-order of the encoded data.
normal	Specifies how the data was processed when it was originally encoded.
radiometric	Specifies how “white runs” are decoded.

Returns

The `XieDecodeG42DParam` structure.

Description

`XieDecodeG42D()` allocates and returns a pointer to an `XieDecodeG42DParam` structure. The returned structure represents the list of parameters required by the decode technique and may be used as the *decode_param* argument of `XieFloImportClientPhoto()` (when the *decode_tech* argument is `xieValDecodeG42D`).

If insufficient memory is available, `XieTecDecodeG42D()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

CCITT-G42D is the CCITT group 3 one dimensional encoding technique as defined by CCITT T.4, "Standardization of Group 3 Facsimile Apparatus for Document Transmission".

encoded_order specifies the bit-order of the encoded data. One of the following standard orientation values can be assigned to *encoded_order*:

```
xieValLSFirst
xieValMSFirst
```

The following table shows the encoded bit order of two bytes (within each byte the first encoded bit is “0”, last is bit “7”):

LSFirst	**MSFirst**
76543210 76543210	01234567 01234567

radiometric specifies that “white runs” in the encoded data should be represented as image ones upon decode (maximum intensity), or conversely, they will be decoded as image zeros if *radiometric* is `False`. *normal* specifies that the data was processed according to its normal fill-order when it was originally encoded.

Structures

`XieDecodeG42D()` sets the structure field `encoded_order` to the value of the argument *`encoded_order`*; the structure field `normal` to the value of the argument *`normal`*; and the structure field `radiometric` to the value of the argument *`radiometric`*.

```
typedef unsigned XieOrientation;
typedef struct {
   XieOrientation encoded_order;
   Bool normal;
   Bool radiometric;
} XieDecodeG42DParam;

/* Definitions of Orientation Types */
#define xieValLSFirst                  1
#define xieValMSFirst                  2
```

See Also

XieFloImportClientPhoto(), XiePutClientData().

Name

XieTecDecodeTIFF2 - allocate and fill an `XieDecodeTIFF2Param` structure.

Synopsis

```
XieDecodeTIFF2Param *XieTecDecodeTIFF2 (encoded_order, normal,
            radiometric)
   XieOrientation encoded_order;
   Bool normal;
   Bool radiometric;
```

Arguments

encoded_order	Specifies the bit-order of the encoded data.
normal	Specifies how the data was processed when it was originally encoded.
radiometric	Specifies how “white runs” are decoded.

Returns

The `XieDecodeTIFF2Param` structure.

Description

`XieDecodeTIFF2()` allocates and returns a pointer to an `XieDecodeTIFF2Param` structure. The returned structure represents the list of parameters required by the decode technique and may be used as the *decode_param* argument of `XieFloImportClientPhoto()` (when the *decode_tech* argument is `xieValDecodeTIFF2`).

If insufficient memory is available, `XieTecDecodeTIFF2()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

TIFF-2 is modified Huffman encoding as described in "TIFF Tag Image File Format", revision 6.0, draft 2, by Aldus Corporation. (TIFF compression scheme 2).

encoded_order specifies the bit-order of the encoded data. One of the following standard orientation values can be assigned to *encoded_order*:

```
xieValLSFirst
xieValMSFirst
```

The following table shows the encoded bit order of two bytes (within each byte the first encoded bit is “0”, last is bit “7”):

LSFirst	**MSFirst**
76543210 76543210	01234567 01234567

radiometric specifies that “white runs” in the encoded data should be represented as image ones upon decode (maximum intensity), or conversely, they will be decoded as image zeros if *radiometric* is `False`. *normal* specifies that the data was processed according to its normal fill-order when it was originally encoded.

Structures

`XieDecodeTIFF2()` sets the structure field `encoded_order` to the value of the argument *`encoded_order`*; the structure field `normal` to the value of the argument *`normal`*; and the structure field `radiometric` to the value of the argument *`radiometric`*.

```
typedef unsigned XieOrientation;
typedef struct {
   XieOrientation encoded_order;
   Bool normal;
   Bool radiometric;
} XieDecodeTIFF2Param;

/* Definitions of Orientation Types */
#define xieValLSFirst                  1
#define xieValMSFirst                  2
```

See Also

XieFloImportClientPhoto(), XiePutClientData().

Name

XieTecDecodeTIFFPackBits - allocate and fill an `XieDecodeTIFFPackBitsParam` structure.

Synopsis

```
XieDecodeTIFFPackBitsParam *XieTecDecodeTIFFPackBits (encoded_order,
            normal)
   XieOrientation encoded_order;
   Bool normal;
```

Arguments

encoded_order	Specifies the bit-order of the encoded data.
normal	Specifies how the data was processed when it was originally encoded.

Returns

The `XieDecodeTIFFPackBitsParam` structure.

Description

`XieDecodeTIFFPackBits()` allocates and returns a pointer to an `XieDecodeTIFFPackBitsParam` structure. The returned structure represents the list of parameters required by the decode technique and may be used as the *decode_param* argument of `XieFloImportClientPhoto()` (when the *decode_tech* argument is `xieValDecodeTIFFPackBits`).

If insufficient memory is available, `XieTecDecodeTIFFPackBits()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

TIFF-PackBits is byte-oriented run-length encoding as described in "TIFF Tag Image File Format", revision 6.0, draft 2, by Aldus Corporation. (TIFF compression scheme 32773).

encoded_order specifies the bit-order of the encoded data. One of the following standard orientation values can be assigned to *encoded_order*:

```
xieValLSFirst
xieValMSFirst
```

The following table shows the encoded bit order of two bytes (within each byte the first encoded bit is “0”, last is bit “7”):

LSFirst	**MSFirst**
76543210 76543210	01234567 01234567

normal specifies that the data was processed according to its normal fill-order when it was originally encoded.

Structures

`XieDecodeTIFFPackBits()` sets the structure field `encoded_order` to the value of the argument *encoded_order*; and the structure field `normal` to the value of the argument *normal*.

```
typedef unsigned XieOrientation;
typedef struct {
    XieOrientation encoded_order;
    Bool normal;
} XieDecodeTIFFPackBitsParam;

/* Definitions of Orientation Types */
#define xieValLSFirst                    1
#define xieValMSFirst                    2
```

See Also

XieFloImportClientPhoto(), XiePutClientData().

Name

XieTecDecodeJPEGBaseline - allocate and fill an `XieDecodeJPEGBaselineParam` structure.

Synopsis

```
XieDecodeJPEGBaselineParam *XieTecDecodeJPEGBaseline (interleave,
            band_order, up_sample)
   XieInterleave interleave;
   XieOrientation band_order;
   Bool up_sample;
```

Arguments

interleave	Specifies how the image bands will be interleaved.
band_order	Specifies the order in which the image bands were originally encoded.
up_sample	Specifies how interleaved encoded data are up-sampled.

Returns

The `XieDecodeJPEGBaselineParam` structure.

Description

`XieDecodeJPEGBaseline()` allocates and returns a pointer to an `XieDecodeJPEGBaselineParam` structure. The returned structure represents the list of parameters required by the decode technique and may be used as the *decode_param* argument of `XieFloImportClientPhoto()` (when the *decode_tech* argument is `xieValDecodeJPEGBaseline`).

If insufficient memory is available, `XieTecDecodeJPEGBaseline()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The JPEG baseline technique is the baseline Huffman DCT encoding technique that is defined in ISO DIS 10918-1 "Digital Compression and Coding of Continuous-tone Still Images". Only JPEG Interchange Format (JIF) is supported: all tables, compressed data, etc. are embedded in the data stream, all delineated by markers.

interleave determines if all bands of a triple band image will be interleaved within a single encoded stream, or if three separate encoded streams will be supplied. One of the following standard interleave values can be assigned to *interleave*:

```
xieValBandByPixel
xieValBandByPlane
```

For triple band data, *band_order* specifies the order in which the image bands were originally encoded. One of the following standard orientation values can be assigned to *band_order*:

```
xieValLSFirst
xieValMSFirst
```

The least significant band of trichromatic data is the first band mentioned in the common name of the colorspace: red is the least significant band of `RGB` data. *band_order* specifies whether this band corresponds with the least significant or most significant image plane. Each plane is decoded into a separate data stream:

band	LSFirst	MSFirst
0	Red band	Blue band
1	Green band	Green band
2	Blue band	Red band

up_sample specifies that if any bands in an interleaved encoded data stream are down-sampled, they should be up-sampled by the JPEG decoder.

The arguments *interleave*, *band_order*, and *up_sample* are ignored for single band images, and *up_sample* is always ignored if *interleave* is band-by-plane. If *up_sample* is `False` and some of the encoded bands of an interleaved image were down-sampled, an alternative method for up-sampling the image would be to use a geometry element with appropriate *band_mask* and *sample* technique parameters.

Structures

`XieDecodeJPEGBaseline()` sets the structure field `interleave` to the value of the argument *interleave*; the structure field `band_order` to the value of the argument *band_order*; and the structure field `up_sample` to the value of the argument *up_sample*.

```
typedef unsigned XieOrientation;
typedef struct {
    XieInterleave interleave;
    XieOrientation band_order;
    Bool up_sample;
} XieDecodeJPEGBaselineParam;
/* Definitions for Interleave */
#define xieValBandByPixel             1
#define xieValBandByPlane             2

/* Definitions of Orientation Types */
#define xieValLSFirst                 1
#define xieValMSFirst                 2
```

See Also

XieFloImportClientPhoto(), XiePutClientData().

Name

XieTecDecodeJPEGLossless - allocate and fill an `XieDecodeJPEGLosslessParam` structure.

Synopsis

```
XieDecodeJPEGLosslessParam *XieTecDecodeJPEGLossless (interleave,
            band_order)
   XieInterleave interleave;
   XieOrientation band_order;
```

Arguments

interleave	Specifies how the image bands will be interleaved.
band_order	Specifies the order in which the image bands were originally encoded.

Returns

The `XieDecodeJPEGLosslessParam` structure.

Description

`XieDecodeJPEGLossless()` allocates and returns a pointer to an `XieDecodeJPEGLosslessParam` structure. The returned structure represents the list of parameters required by the decode technique and may be used as the *decode_param* argument of `XieFloImportClientPhoto()` (when the *decode_tech* argument is `xieValDecodeJPEGLossless`).

If insufficient memory is available, `XieTecDecodeJPEGLossless()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The JPEG lossless technique is the Huffman predictive sequential lossless encoding technique that is defined in ISO DIS 10918-1 "Digital Compression and Coding of Continuous-tone Still Images". This technique is not available in the R6 sample implementation of XIE.

interleave describes how the bands of triple band data are interleaved; either all bands are interleaved within a single encoded stream, or three separate encoded streams are expected. One of the following standard interleave values can be assigned to *interleave*:

```
xieValBandByPixel
xieValBandByPlane
```

For triple band data, *band_order* specifies the order in which the image bands were originally encoded. One of the following standard orientation values can be assigned to *band_order*:

```
xieValLSFirst
xieValMSFirst
```

The least significant band of trichromatic data is the first band mentioned in the common name of the colorspace: red is the least significant band of `RGB` data. *band_order* specifies whether this band corresponds with the least significant or most significant image plane. Each plane is decoded into a separate data stream:

band	LSFirst	MSFirst
0	Red band	Blue band
1	Green band	Green band
2	Blue band	Red band

The arguments *interleave* and *band_order* are ignored for single band images.

Structures

XieDecodeJPEGLossless() sets the structure field interleave to the value of the argument *interleave*; and the structure field band_order to the value of the argument *band_order*.

```
typedef unsigned XieOrientation;
typedef struct {
    XieInterleave interleave;
    XieOrientation band_order;
} XieDecodeJPEGLosslessParam;
/* Definitions for Interleave */
#define xieValBandByPixel              1
#define xieValBandByPlane              2

/* Definitions of Orientation Types */
#define xieValLSFirst                  1
#define xieValMSFirst                  2
```

See Also

XieFloImportClientPhoto(), XiePutClientData().

Name

XieTecDitherOrderedParam - allocate and fill an `XieDitherOrderedParam` structure.

Synopsis

```
XieDitherOrderedParam *XieTecDitherOrderedParam (threshold_order)
   unsigned int threshold_order;
```

Arguments

threshold_order Specifies a value to determine the size of the dither matrix.

Returns

The `XieDitherOrderedParam` structure.

Description

`XieDitherOrdered()` allocates and returns a pointer to an `XieDitherOrderedParam` structure. The returned structure represents the list of parameters required by the decode technique and may be used as the *decode_param* argument of `XieFloImportClientPhoto()` (when the *decode_tech* argument is `xieValDitherOrdered`).

If insufficient memory is available, `XieTecDitherOrdered()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The dispersed-dot ordered dither technique replaces a matrix, or block, of pixels with a patterned matrix of pixels. This patterned matrix of pixels is applied across the entire image. Since these patterns may introduce artifacts that are distracting to the eye, the *threshold_order* parameter is available to determine the size of the dither matrix, and therefore, the number of levels that can be simulated. If the value of *threshold_order* is m, the threshold matrix can simulate 2^m+1 levels.

Structures

`XieDitherOrdered()` sets the structure field `threshold_order` to the value of the argument *threshold_order*.

```
typedef struct {
    unsigned int threshold_order;
} XieDitherOrderedParam;
```

See Also

XieFloDither().

Name

XieTecEncodeUncompressedSingle - allocate and fill an `XieEncodeUncompressedSingleParam` structure.

Synopsis

```
XieEncodeUncompressedSingleParam *XieTecEncodeUncompressedSingle
            (fill_order, pixel_order, pixel_stride, scanline_pad)
   XieOrientation fill_order;
   XieOrientation pixel_order;
   unsigned int pixel_stride;
   unsigned int scanline_pad;
```

Arguments

fill_order	Specifies the method of pixel packing .
pixel_order	Specifies pixel ordering within the data stream.
pixel_stride	Specifies the number of bits between consecutive pixels within a scanline.
scanline_pad	Specifies a multiple of bytes to which each scanline is padded.

Returns

The `XieEncodeUncompressedSingleParam` structure.

Description

`XieEncodeUncompressedSingle()` allocates and returns a pointer to an `XieEncodeUncompressedSingleParam` structure. The returned structure represents the list of parameters required by the encode technique and may be used as the *encode_param* argument of `XieFloExportClientPhoto()` and `XieFloExportPhotomap()` (when the *encode_tech* argument is `xieValEncodeUncompressedSingle`).

If insufficient memory is available, `XieTecEncodeUncompressedSingle()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The encode uncompressed single technique is used when no compression scheme is to be applied to single band image data. The parameters define the format of the data stream of uncompressed data that is made available for client retrieval via `XieGetClientData()`. When multiple pixels are put in the same byte or a pixel spans multiple bytes, *fill_order* specifies whether the pixels (or parts of pixels) are packed into the most or least significant bits of a byte first. For pixels that span a byte boundary, *pixel_order* defines whether the most or least significant bits of the pixel are put first within the data stream.

One of the following standard orientation values can be assigned to *fill_order* and *pixel_order*:

```
xieValLSFirst
xieValMSFirst
```

The table below shows the relationship between *fill_order and pixel_order*, using two 10-bit pixels, each with two bits of pad (within each pixel the LS-bits are “0” and “a”, the MS-bits are “9” and “j”, the pad bits are “p”).

fill-order	LSFirst (pixel-order)	MSFirst (pixel-order)
LSFirst	76543210 dcbapp98 ppjihgfe	98765432 jihgpp10 ppfedcba
MSFirst	76543210 98ppdcba jihgfepp	98765432 10ppjihg fedcbapp

pixel_stride is the number of bits between the start of consecutive pixels within a scanline. It must be at least enough bits to contain the number of source levels. *scanline_pad* defines a multiple of bytes to which each scanline is padded; valid values are: 0 (not aligned), 1, 2, 4, 8, and 16.

Structures

XieEncodeUncompressedSingle() sets the structure field fill_order to the value of the argument *fill_order*; the structure field pixel_order to the value of the argument *pixel_order*; the structure field pixel_stride to the value of the argument *pixel_stride*; and the structure field scanline_pad to the value of the argument *scanline_pad*.

```
typedef unsigned XieOrientation;
typedef struct {
   XieOrientation fill_order;
   XieOrientation pixel_order;
   unsigned int pixel_stride;
   unsigned int scanline_pad;
} XieEncodeUncompressedSingleParam;

/* Definitions of Orientation Types */
#define xieValLSFirst                  1
#define xieValMSFirst                  2
```

See Also

XieFloExportClientPhoto(), XieFloExportPhotomap(), XieGetClientData().

Name

XieTecEncodeUncompressedTriple - allocate and fill an `XieEncodeUncompressedTripleParam` structure.

Synopsis

```
XieEncodeUncompressedTripleParam *XieTecEncodeUncompressedTriple
            (fill_order, pixel_order, band_order, interleave,
            pixel_stride, scanline_pad)
   XieOrientation fill_order;
   XieOrientation pixel_order;
   XieOrientation band_order;
   XieInterleave interleave;
   unsigned char pixel_stride[3];
   unsigned char scanline_pad[3];
```

Arguments

fill_order	Specifies the method of pixel packing .
pixel_order	Specifies pixel ordering within the data stream.
band_order	Specifies the order of the image data sent through the protocol stream.
interleave	Specifies how the image bands are interleaved.
pixel_stride	Specifies the number of bits between consecutive pixels within a scanline.
scanline_pad	Specifies a multiple of bytes to which each scanline is padded.

Returns

The `XieEncodeUncompressedTripleParam` structure.

Description

`XieEncodeUncompressedTriple()` allocates and returns a pointer to an `XieEncodeUncompressedTripleParam` structure. The returned structure represents the list of parameters required by the encode technique and may be used as the *encode_param* argument of `XieFloExportClientPhoto()` and `XieFloExportPhotomap()` (when the *encode_tech* argument is `xieValEncodeUncompressedTriple`).

If insufficient memory is available, `XieTecEncodeUncompressedTriple()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The encode uncompressed triple technique is used when no compression scheme is to be applied to triple band image data. The parameters define the format of the data stream of uncompressed data that is made available for client retrieval via `XieGetClientData()`. When multiple pixels are put in the same byte or a pixel spans multiple bytes, *fill_order* specifies whether the pixels (or parts of pixels) are packed into the most or least significant bits of a byte first. For pixels that span a byte boundary, *pixel_order* defines whether the most or least significant bits of the pixel are put first within the data stream.

One of the following standard orientation values can be assigned to *fill_order* and *pixel_order*:

```
xieValLSFirst
xieValMSFirst
```

The table below shows the relationship between *fill_order and pixel_order*, using two 10-bit pixels, each with two bits of pad (within each pixel the LS-bits are “0” and “a”, the MS-bits are “9” and “j”, the pad bits are “p”)

fill-order	LSFirst (pixel-order)	MSFirst (pixel-order)
LSFirst	76543210 dcbapp98 ppjihgfe	98765432 jihgpp10 ppfedcba
MSFirst	76543210 98ppdcba jihgfepp	98765432 10ppjihg fedcbapp

pixel_stride is the number of bits between the start of consecutive pixels within a scanline. It must be at least enough bits to contain the number of source levels. *scanline_pad* defines a multiple of bytes to which each scanline is padded; valid values are: 0 (not aligned), 1, 2, 4, 8, and 16. *band_order* is the order of the image bands or image planes as they are transmitted through the protocol stream. One of the following standard orientation values can be assigned to *band_order*:

```
xieValLSFirst
xieValMSFirst
```

The least significant band of trichromatic data is the first band mentioned in the common name of the colorspace: red is the least significant band of RGB data. For band-by-pixel data, *band_order* specifies whether this band is put in the least or most significant bits of a pixel:

LSFirst	MSFirst
$B_1B_0G_2G_1G_0R_2R_1R_0$	$R_2R_1R_0G_2G_1G_0B_1B_0$

For band-by-plane data, *band_order* specifies whether this band corresponds with the least significant or most significant image plane. Each plane is transported as a separate data stream:

band	LSFirst	MSFirst
0	$R_7R_6R_5R_4R_3R_2R_1R_0$	$B_7B_6B_5B_4B_3B_2B_1B_0$
1	$G_7G_6G_5G_4G_3G_2G_1G_0$	$G_7G_6G_5G_4G_3G_2G_1G_0$
2	$B_7B_6B_5B_4B_3B_2B_1B_0$	$R_7R_6R_5R_4R_3R_2R_1R_0$

interleave describes how the bands are interleaved (per pixel within a single plane, or sent as three separate planes). One of the following standard interleave values can be assigned to *interleave*:

```
xieValBandByPixel
xieValBandByPlane
```

Export of down-sampled band-by-pixel data is not supported: all bands must have equal widths and equal heights.

Structures

`XieEncodeUncompressedTriple()` sets the structure field `fill_order` to the value of the argument *fill_order*; the structure field `pixel_order` to the value of the argument *pixel_order*; the structure field `band_order` to the value of the argument *band_order*; the structure field `interleave` to the value of the argument *`interleave`*; the structure field `pixel_stride` to the value of the argument *`pixel_stride`*; and the structure field `scanline_pad` to the value of the argument *`scanline_pad`*.

```
typedef unsigned XieOrientation;
typedef struct {
   unsigned char pixel_stride[3];
   XieOrientation pixel_order;
   unsigned char scanline_pad[3];
   XieOrientation fill_order;
   XieOrientation band_order;
   XieInterleave interleave;
} XieEncodeUncompressedTripleParam;

/* Definitions of Orientation Types */
#define xieValLSFirst                   1
#define xieValMSFirst                   2

/* Definitions for Interleave */
#define xieValBandByPixel               1
#define xieValBandByPlane               2
```

See Also

XieFloExportClientPhoto(), XieFloExportPhotomap(), XieGetClientData().

Name

XieTecEncodeG31D - allocate and fill an `XieEncodeG31DParam` structure.

Synopsis

```
XieEncodeG31DParam *XieTecEncodeG31D (align_eol, radiometric,
            encoded_order)
   Bool align_eol;
   Bool radiometric;
   XieOrientation encoded_order;
```

Arguments

align_eol	Specifies the use of fill bits preceding EOL codes.
radiometric	Specifies how “white runs” are encoded.
encoded_order	Specifies the bit-order of the encoded data.

Returns

The `XieEncodeG31DParam` structure.

Description

`XieEncodeG31D()` allocates and returns a pointer to an `XieEncodeG31DParam` structure. The returned structure represents the list of parameters required by the encode technique and may be used as the *encode_param* argument of `XieFloExportClientPhoto()` and `XieFloExportPhotomap()` (when the *encode_tech* argument is `xieValEncodeG31D`).

If insufficient memory is available, `XieTecEncodeG31D()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

CCITT-G31D is the CCITT group 3 one dimensional encoding technique as defined by CCITT T.4, "Standardization of Group 3 Facsimile Apparatus for Document Transmission".

If `True`, *align_eol*, specifies that sufficient fill bits must precede EOL codes to guarantee that each EOL will end on a byte boundary (thus EOL will be a nibble of 0 followed by a byte of 1: $xxxx,0000_2$ $0000,0001_2$). *radiometric* specifies that image ones will be encoded as “white runs”, or conversely, image zeros will be encoded as “white runs”, if *radiometric* is `False`. *encoded_order* specifies the bit-order for the encoded data. One of the following standard orientation values can be assigned to *encoded_order*:

```
xieValLSFirst
xieValMSFirst
```

The following table shows the encoded bit order of two bytes (within each byte the first encoded bit is “0”, last is bit “7”):

LSFirst	MSFirst
76543210 76543210	01234567 01234567

Structures

`XieEncodeG31D()` sets the structure field `align_eol` to the value of the argument *`align_eol`*; the structure field `radiometric` to the value of the argument *`radiometric`*; and the structure field `encoded_order` to the value of the argument *`encoded_order`*.

```
typedef unsigned XieOrientation;
typedef struct {
   Bool align_eol;
   Bool radiometric;
   XieOrientation encoded_order;
} XieEncodeG31DParam;

/* Definitions of Orientation Types */
#define xieValLSFirst                    1
#define xieValMSFirst                    2
```

See Also

XieFloExportClientPhoto(), XieFloExportPhotomap(), XieGetClientData().

Name

XieTecEncodeG32D - allocate and fill an `XieEncodeG32DParam` structure.

Synopsis

```
XieEncodeG32DParam *XieTecEncodeG32D (uncompressed, align_eol,
            radiometric, encoded_order, k_factor)
   Bool uncompressed;
   Bool align_eol;
   Bool radiometric;
   XieOrientation encoded_order;
   unsigned long k_factor;
```

Arguments

uncompressed	Specifies the use of the `uncompressed-mode` CCITT extension.
align_eol	Specifies the use of fill bits preceding EOL codes.
radiometric	Specifies how “white runs” are encoded.
encoded_order	Specifies the bit-order of the encoded data.
k_factor	Specifies the number of 2D scanlines to produce for each 1D scanline.

Returns

The `XieEncodeG32DParam` structure.

Description

`XieEncodeG32D()` allocates and returns a pointer to an `XieEncodeG32DParam` structure. The returned structure represents the list of parameters required by the encode technique and may be used as the *encode_param* argument of `XieFloExportClientPhoto()` and `XieFloExportPhotomap()` (when the *encode_tech* argument is `xieValEncodeG32D`).

If insufficient memory is available, `XieTecEncodeG32D()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

CCITT-G32D is the CCITT group 3 two dimensional encoding technique as defined by CCITT T.4, "Standardization of Group 3 Facsimile Apparatus for Document Transmission".

If `True`, *uncompressed*, will enable the use of the uncompressed-mode CCITT extension. If `True`, *align_eol*, specifies that sufficient fill bits must precede EOL codes to guarantee that each EOL will end on a byte boundary (thus EOL will be a nibble of 0 followed by a byte of 1: $xxxx,0000_2$ $0000,0001_2$). *radiometric* specifies that image ones will be encoded as “white runs”, or conversely, image zeros will be encoded as “white runs”, if *radiometric* is `False`. *encoded_order* specifies the bit-order for the encoded data. One of the following standard orientation values can be assigned to *encoded_order*:

```
xieValLSFirst
xieValMSFirst
```

The following table shows the encoded bit order of two bytes (within each byte the first encoded bit is “0”, last is bit “7”):

LSFirst	MSFirst
76543210 76543210	01234567 01234567

k_factor specifies the number of 2-D scanlines to produce for each 1-D scanline.

Structures

`XieEncodeG32D()` sets the structure field `uncompressed` to the value of the argument *uncompressed*; the structure field `align_eol` to the value of the argument *align_eol*; the structure field `radiometric` to the value of the argument *radiometric*; the structure field `encoded_order` to the value of the argument *encoded_order*; and the structure field `k_factor` to the value of the argument *k_factor*.

```
typedef unsigned XieOrientation;
typedef struct {
   Bool uncompressed;
   Bool align_eol;
   Bool radiometric;
   XieOrientation encoded_order;
   unsigned long k_factor;
} XieEncodeG32DParam;

/* Definitions of Orientation Types */
#define xieValLSFirst                  1
#define xieValMSFirst                  2
```

See Also

XieFloExportClientPhoto(), XieFloExportPhotomap(), XieGetClientData().

Name

XieTecEncodeG42D - allocate and fill an `XieEncodeG42DParam` structure.

Synopsis

```
XieEncodeG42DParam *XieTecEncodeG42D (uncompressed, radiometric,
            encoded_order)
   Bool uncompressed;
   Bool radiometric;
   XieOrientation encoded_order;
```

Arguments

uncompressed	Specifies the use of the `uncompressed-mode` CCITT extension.
radiometric	Specifies how “white runs” are encoded.
encoded_order	Specifies the bit-order of the encoded data.

Returns

The `XieEncodeG42DParam` structure.

Description

`XieEncodeG42D()` allocates and returns a pointer to an `XieEncodeG42DParam` structure. The returned structure represents the list of parameters required by the encode technique and may be used as the *encode_param* argument of `XieFloExportClientPhoto()` and `XieFloExportPhotomap()` (when the *encode_tech* argument is `xieValEncodeG42D`).

If insufficient memory is available, `XieTecEncodeG42D()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

CCITT-G42D is the CCITT group 4 two dimensional encoding technique as defined by CCITT T.6, "Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus".

If `True`, *uncompressed*, will enable the use of the uncompressed-mode CCITT extension. *radiometric* specifies that image ones will be encoded as “white runs”, or conversely, image zeros will be encoded as “white runs”, if *radiometric* is `False`. *encoded_order* specifies the bit-order for the encoded data. One of the following standard orientation values can be assigned to *encoded_order*:

```
xieValLSFirst
xieValMSFirst
```

The following table shows the encoded bit order of two bytes (within each byte the first encoded bit is “0”, last is bit “7”):

LSFirst	MSFirst
76543210 76543210	01234567 01234567

Structures

`XieEncodeG42D()` sets the structure field `uncompressed` to the value of the argument *`uncompressed`*; the structure field `radiometric` to the value of the argument *`radiometric`*; and the structure field `encoded_order` to the value of the argument *`encoded_order`*.

```
typedef unsigned XieOrientation;
typedef struct {
   Bool uncompressed;
   Bool radiometric;
   XieOrientation encoded_order;
} XieEncodeG42DParam;

/* Definitions of Orientation Types */
#define xieValLSFirst                  1
#define xieValMSFirst                  2
```

See Also

XieFloExportClientPhoto(), XieFloExportPhotomap(), XieGetClientData().

Name

XieTecEncodeServerChoice - allocate and fill an `XieEncodeServerChoiceParam` structure.

Synopsis

```
XieEncodeServerChoiceParam *XieTecEncodeServerChoice (preference)
   unsigned int preference;
```

Arguments

preference Specifies a "hint" to help the server make its choice.

Returns

The `XieEncodeServerChoiceParam` structure.

Description

`XieEncodeServerChoice()` allocates and returns a pointer to an `XieEncodeServerChoiceParam` structure. The returned structure represents the list of parameters required by the encode technique and may be used as the *encode_param* argument of `XieFloExportClientPhoto()` and `XieFloExportPhotomap()` (when the *encode_tech* argument is `xieValEncodeServerChoice`).

If insufficient memory is available, `XieTecEncodeServerChoice()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The server choice technique allows the server to choose an encode technique when exporting to a photomap. A photomap is a server resource which can be used to store image data. *preference* specifies an optional "hint" that can be provided to help the server make its choice, but the server is not obligated to obey the hint. One of the following standard server choice preference values can be assigned to *preference*:

```
xieValPreferDefault
xieValPreferSpace
xieValPreferTime
```

`xieValPreferTime` hints that retrieval performance is the desired metric, whereas `xieValPreferSpace` indicates that frugal use of storage space is more important. Normally the server choice technique must choose a lossless encode technique, but when data is received from an adjacent upstream import element it may choose to pass the import element's input data directly to the photomap.

Structures

`XieEncodeServerChoice()` sets the structure field `preference` to the value of the argument *preference*.

```
typedef struct {
   unsigned int preference;
} XieEncodeServerChoiceParam;

/* Definitions for ServerChoice Preference Hints */
#define xieValPreferDefault            0
```

```
#define xieValPreferSpace                1
#define xieValPreferTime                 2
```

See Also

XieFloExportPhotomap().

Name

XieTecEncodeJPEGBaseline - allocate and fill an `XieEncodeJPEGBaselineParam` structure.

Synopsis

```
XieEncodeJPEGBaselineParam *XieTecEncodeJPEGBaseline (interleave,
            band_order, horizontal_samples, vertical_samples,
            q_table, q_size, ac_table, ac_size, dc_table, dc_size)
   XieInterleave interleave;
   XieOrientation band_order;
   unsigned char horizontal_samples[3];
   unsigned char vertical_samples[3];
   char *q_table;
   unsigned int q_size;
   char *ac_table;
   unsigned int ac_size;
   char *dc_table;
   unsigned int dc_size;
```

Arguments

interleave	Specifies how the image bands will be interleaved.
band_order	Specifies the order in which the image bands were originally encoded.
horizontal_samples	Specifies the horizontal sampling factor.
vertical_samples	Specifies the vertical sampling factor.
q_table	Specifies the quantization table.
q_size	Specifies the number of elements in *q_table*.
ac_table	Specifies the AC Huffman table.
ac_size	Specifies the number of elements in *ac_table*.
dc_table	Specifies the DC Huffman table.
dc_size	Specifies the number of elements in *dc_table*.

Returns

The `XieEncodeJPEGBaselineParam` structure.

Description

`XieEncodeJPEGBaseline()` allocates and returns a pointer to an `XieEncodeJPEGBaselineParam` structure. The returned structure represents the list of parameters required by the encode technique and may be used as the *encode_param* argument of `XieFloExportClientPhoto()` and `XieFloExportPhotomap()` (when the *encode_tech* argument is `xieValEncodeJPEGBaseline`).

If insufficient memory is available, `XieTecEncodeJPEGBaseline()` returns `NULL`. `XieTecEncodeJPEGBaseline()` allocates memory for the structure fields `q_table`, `ac_table`, and `dc_table`; to free the memory allocated to the structure `XieEncodeJPEGBaselineParam`, use `XieFreeEncodeJPEGBaseline()`.

JPEG-Baseline is baseline Huffman DCT encoding as defined by ISO DIS 10918-1 "Digital Compression and Coding of Continuous-tone Still Images". A stream of JPEG Interchange Format

(JIF) data is produced: all tables, compressed data, etc. are embedded in the data stream, all delineated by markers.

interleave determines if all bands of a triple band image will be interleaved within a single encoded stream, or if three separate encoded streams will be produced. One of the following standard interleave values can be assigned to *interleave*:

```
xieValBandByPixel
xieValBandByPlane
```

For triple band data, *band_order* specifies the order in which the image bands were originally encoded. One of the following standard orientation values can be assigned to *band_order*:

```
xieValLSFirst
xieValMSFirst
```

The least significant band of trichromatic data is the first band mentioned in the common name of the colorspace: red is the least significant band of RGB data. *band_order* specifies whether this band corresponds with the least significant or most significant image plane. Each plane is encoded from a separate data stream:

band	**LSFirst**	**MSFirst**
0	Red band	Blue band
1	Green band	Green band
2	Blue band	Red band

The arguments *horizontal_samples* and *vertical_samples* are the horizontal and vertical sampling factors. *q_table* is the quantization table. *ac_table* specifies the AC Huffman table and *dc_table* specifies the DC Huffman table.

There may be one *q_table* per band or a single *q_table* shared between all bands. Generally there is a single AC/DC pair of Huffman tables, but for triple band band-by-pixel data there may be two pairs of tables (one for the luminance band and the other for the chromanance bands). If any table is specified with zero length, it indicates that the server implementor is to supply that table (e.g. the example tables defined in ISO DIS 10918-1 "Digital Compression and Coding of Continuous-tone Still Images").

The arguments *interleave*, *band_order*, *horizontal_samples*, and *vertical_samples* are ignored for single band images, and *horizontal_samples* and *vertical_samples* are always ignored if *interleave* is band-by-plane. *horizontal_samples* and *vertical_samples* share the definitions and restrictions stipulated for parameters H_i and V_i, respectively, that are specified in annexes A and B of ISO DIS 10918-1 "Digital Compression and Coding of Continuous-tone Still Images".

Structures

`XieEncodeJPEGBaseline()` sets the structure field `interleave` to the value of the argument *interleave*; the structure field `band_order` to the value of the argument *band_order*; the structure field `horizontal_sample` to the values of the argument *horizontal_sample*; the structure field `vertical_sample` to the values of the argument *vertical_sample*; and the structure fields `q_table, q_size, ac_table, ac_size, dc_table, dc_size` to the values of the arguments *q_table, q_size, ac_table, ac_size, dc_table, dc_size*.

```
typedef unsigned XieOrientation;
typedef struct {
   XieInterleave interleave;
   XieOrientation band_order;
   unsigned char horizontal_samples[3];
   unsigned char vertical_samples[3];
   char *q_table;
   unsigned int q_size;
   char *ac_table;
   unsigned int ac_size;
   char *dc_table;
   unsigned int dc_size;
} XieEncodeJPEGBaselineParam;

/* Definitions for Interleave */
#define xieValBandByPixel              1
#define xieValBandByPlane              2

/* Definitions of Orientation Types */
#define xieValLSFirst                  1
#define xieValMSFirst                  2
```

See Also

XieFloExportClientPhoto(), XieFloExportPhotomap(), XieFreeEncodeJPEGBaseline(), XieGetClientData().

Name

XieTecEncodeJPEGLossless - allocate and fill an `XieEncodeJPEGLosslessParam` structure.

Synopsis

```
XieEncodeJPEGLosslessParam *XieTecEncodeJPEGLossless (interleave,
            band_order, predictor, table, table_size)
   XieInterleave interleave;
   XieOrientation band_order;
   unsigned char predictor[3];
   char *table;
   unsigned int table_size;
```

Arguments

interleave	Specifies how the image bands will be interleaved.
band_order	Specifies the order in which the image bands were originally encoded.
predictor	Specifies the predictor selection value.
table	Specifies the lossless entropy encoding table.
table_size	Specifies the number of elements in *table*.

Returns

The `XieEncodeJPEGLosslessParam` structure.

Description

`XieEncodeJPEGLossless()` allocates and returns a pointer to an `XieEncodeJPEGLosslessParam` structure. The returned structure represents the list of parameters required by the encode technique and may be used as the *encode_param* argument of `XieFloExportClientPhoto()` and `XieFloExportPhotomap()` (when the *encode_tech* argument is `xieValEncodeJPEGLossless`).

If insufficient memory is available, `XieTecEncodeJPEGLossless()` returns `NULL`. `XieTecEncodeJPEGLossless()` allocates memory for the structure field `table`; to free the memory allocated to the structure `XieEncodeJPEGLosslessParam`, use `XieFreeEncodeJPEGLossless()`.

JPEG-Lossless, corresponds to frames encoded using Huffman, predictive sequential lossless encoding as defined by ISO DIS 10918-1 "Digital Compression and Coding of Continuous-tone Still Images". A data stream of JPEG Interchange Format (JIF) data is returned: all tables, compressed data, etc. are embedded in the data stream, all delineated by markers.
This technique is not available in the R6 sample implementation of XIE.

interleave determines if all bands of a triple band image will be interleaved within a single encoded stream, or if three separate encoded streams will be produced. One of the following standard interleave values can be assigned to *interleave*:

```
xieValBandByPixel
xieValBandByPlane
```

For triple band data, *band_order* specifies the order in which the image bands were originally encoded. One of the following standard orientation values can be assigned to *band_order*:

```
xieValLSFirst
xieValMSFirst
```

The least significant band of trichromatic data is the first band mentioned in the common name of the colorspace: red is the least significant band of RGB data. *band_order* specifies whether this band corresponds with the least significant or most significant image plane. Each plane is encoded from a separate data stream:

band	**LSFirst**	**MSFirst**
0	Red band	Blue band
1	Green band	Green band
2	Blue band	Red band

The arguments *interleave* and *band_order* are ignored for single band images.

predictor is the predictor selection value (one per band). *table* is the lossless entropy encoding table (up to one per band). Specifying a *table* of length zero indicates that the server implementor should supply a table (e.g. the example tables defined in ISO DIS 10918-1 "Digital Compression and Coding of Continuous-tone Still Images").

Structures

`XieEncodeJPEGLossless()` sets the structure field `interleave` to the value of the argument *interleave*; the structure field `band_order` to the value of the argument *band_order*; the structure field `predictor` to the values of the argument *predictor*; and the structure fields `table, table_size` to the values of the arguments *table*, *table_size*.

```
typedef unsigned XieOrientation;
typedef struct {
   XieInterleave interleave;
   XieOrientation band_order;
   unsigned char predictor[3];
   char *table;
   unsigned int table_size;
} XieEncodeJPEGLosslessParam;
/* Definitions for Interleave */
#define xieValBandByPixel             1
#define xieValBandByPlane             2

/* Definitions of Orientation Types */
#define xieValLSFirst                 1
#define xieValMSFirst                 2
```

See Also

XieFloExportClientPhoto(), XieFloExportPhotomap(), XieGetClientData().

Name

XieTecEncodeTIFF2 - allocate and fill an `XieEncodeTIFF2Param` structure.

Synopsis

```
XieEncodeTIFF2Param *XieTecEncodeTIFF2 (encoded_order, radiometric)
   XieOrientation encoded_order;
   Bool radiometric;
```

Arguments

encoded_order	Specifies the bit-order of the encoded data.
radiometric	Specifies how “white runs” are decoded.

Returns

The `XieEncodeTIFF2Param` structure.

Description

`XieEncodeTIFF2()` allocates and returns a pointer to an `XieEncodeTIFF2Param` structure. The returned structure represents the list of parameters required by the encode technique and may be used as the *encode_param* argument of `XieFloExportClientPhoto()` and `XieFloExportPhotomap()` (when the *encode_tech* argument is `xieValEncodeTIFF2`).

If insufficient memory is available, `XieTecEncodeTIFF2()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

TIFF-2 is modified Huffman compression as described in "TIFF Tag Image File Format", revision 6.0, draft 2, by Aldus Corporation. (TIFF compression scheme 2).

radiometric specifies that image ones will be encoded as “white runs”, or conversely, image zeros will be encoded as “white runs”, if *radiometric* is `False`. *encoded_order* specifies the bit-order for the encoded data. One of the following standard orientation values can be assigned to *encoded_order*:

```
xieValLSFirst
xieValMSFirst
```

The following table shows the encoded bit order of two bytes (within each byte the first encoded bit is “0”, last is bit “7”):

LSFirst	**MSFirst**
76543210 76543210	01234567 01234567

Structures

`XieEncodeTIFF2()` sets the structure field `encoded_order` to the value of the argument *encoded_order*; and the structure field `radiometric` to the value of the argument *radiometric*.

```
typedef unsigned XieOrientation;
typedef struct {
```

```
   XieOrientation encoded_order;
   Bool radiometric;
} XieEncodeTIFF2Param;

/* Definitions of Orientation Types */
#define xieValLSFirst                  1
#define xieValMSFirst                  2
```

See Also

XieFloExportClientPhoto(), XieFloExportPhotomap(), XieGetClientData().

Name

XieTecEncodeTIFFPackBits - allocate and fill an `XieEncodeTIFFPackBitsParam` structure.

Synopsis

```
XieEncodeTIFFPackBitsParam *XieTecEncodeTIFFPackBits (encoded_order)
   XieOrientation encoded_order;
```

Arguments

encoded_order	Specifies the bit-order of the encoded data.

Returns

The `XieEncodeTIFFPackBitsParam` structure.

Description

`XieEncodeTIFFPackBits()` allocates and returns a pointer to an `XieEncodeTIFFPackBitsParam` structure. The returned structure represents the list of parameters required by the encode technique and may be used as the *encode_param* argument of `XieFloExportClientPhoto()` and `XieFloExportPhotomap()` (when the *encode_tech* argument is `xieValEncodeTIFFPackBits`).

If insufficient memory is available, `XieTecEncodeTIFFPackBits()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

TIFF-PackBits is byte-oriented run-length encoding as described in "TIFF Tag Image File Format", revision 6.0, draft 2, by Aldus Corporation. (TIFF compression scheme 32773).

encoded_order specifies the bit-order of the encoded data. One of the following standard orientation values can be assigned to *encoded_order*:

```
xieValLSFirst
xieValMSFirst
```

The following table shows the encoded bit order of two bytes (within each byte the first encoded bit is “0”, last is bit “7”):

LSFirst	**MSFirst**
76543210 76543210	01234567 01234567

Structures

`XieEncodeTIFFPackBits()` sets the structure field `encoded_order` to the value of the argument *encoded_order*.

```
typedef unsigned XieOrientation;
typedef struct {
   XieOrientation encoded_order;
} XieEncodeTIFFPackBitsParam;

/* Definitions of Orientation Types */
```

```
#define xieValLSFirst                  1
#define xieValMSFirst                  2
```

See Also

XieFloExportClientPhoto(), XieFloExportPhotomap(), XieGetClientData().

Name

XieTecGeomAntialiasByArea - allocate and fill an `XieGeomAntialiasByAreaParam` structure.

Synopsis

```
XieGeomAntialiasByAreaParam *XieTecGeomAntialiasByArea (simple)
   int simple;
```

Arguments

simple	Specifies the approximation form to use.

Returns

The `XieGeomAntialiasByAreaParam` structure.

Description

`XieGeomAntialiasByArea()` allocates and returns a pointer to an `XieGeomAntialiasByAreaParam` structure. The returned structure represents the list of parameters required by the geometry technique and may be used as the *sample_param* argument of `XieFloGeometry()` (when the *sample_tech* argument is `xieValGeomAntialiasByArea`).

If insufficient memory is available, `XieTecGeomAntialiasByArea()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

Anti-aliasing techniques incorporate information from an "area" of pixels in the input image in order to produce each output pixel. This implies that line dropouts and other artifacts will occur less often, and the output image may have markedly better resemblance to the input image. The technique AntialiasByArea computes the output image by assigning to each output pixel the weighted average of the intensity values of input pixels which fall within its “area”. That is, the four corners of the output pixel are projected back onto the input image.

If *simple* is zero (0), the size and shape of the "area" are determined by the parameters of the geometric transformation. The boundaries of the "area" may not fall on pixel boundaries and, in the case of non-orthogonal rotation of the image, "area" may not be rectangular. Partial input pixel values may have to be calculated: the antialias-by-area technique preserves shape but can be very slow computationally.

Because of the computational complexity of this method, two approximations are supported. If *simple* is non-zero, the pixels covered by the projected area will be averaged without regard to the relative amount of area which they contain: if they are touched by the area, they are included in a simple average. If *simple* is set to *N*, with *N* odd and greater than one (3,5,7, ...), then only the center point of the output pixel is projected, and a simple average is taken of an *N* by *N* window centered on the projection. For best results, *N* should correspond roughly to the amount of scaling which will be done.

Structures

`XieGeomAntialiasByArea()` sets the structure field `simple` to the value of the argument *simple*.

```
typedef struct {
    int simple;
} XieGeomAntialiasByAreaParam;
```

See Also

XieFloGeometry().

Name

XieTecGeomAntialiasByLowpass - allocate and fill an `XieGeomAntialiasByLowpassParam` structure.

Synopsis

```
XieGeomAntialiasByLowpassParam *XieTecGeomAntialiasByLowpass
            (kernel_size)
   int kernel_size;
```

Arguments

kernel_size Specifies the size of the image kernel.

Returns

The `XieGeomAntialiasByLowpassParam` structure.

Description

`XieGeomAntialiasByLowpass()` allocates and returns a pointer to an `XieGeomAntialiasByLowpassParam` structure. The returned structure represents the list of parameters required by the geometry technique and may be used as the *sample_param* argument of `XieFloGeometry()` (when the *sample_tech* argument is `xieValGeomAntialiasByLowpass`).

If insufficient memory is available, `XieTecGeomAntialiasByLowpass()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

Anti-aliasing techniques incorporate information from an "area" of pixels in the input image in order to produce each output pixel. This implies that line dropouts and other artifacts will occur less often, and the output image may have markedly better resemblance to the input image. The technique AntialiasByLowpass represents an approximation to antialias-by-area which can be faster, yet provides similar results. First, a low-pass filter is applied by passing an *n*x*n* boxcar kernel over the original input image. Output pixel values are then derived using a nearest neighbor sampling method that selects the value of the input pixel in the upper-left corner of the area mapped back from the output pixel.

The user is allowed to select the size of the image kernel via the *kernel_size* parameter. For best results, *kernel_size* should be chosen to correspond roughly to the amount of scaling which will be done. Note that the server chooses the best kernel for the appropriate size; the values used in the kernel are not alterable by the client application. Clients wishing to specify the kernel in more detail should use the convolve technique directly.

Structures

`XieGeomAntialiasByLowpass()` sets the structure field `kernel_size` to the value of the argument *kernel_size*.

```
typedef struct {
   int kernel_size;
} XieGeomAntialiasByLowpassParam;
```

See Also

XieFloGeometry(), XieFloConvolve(), XieTecGeomAntialiasByArea().

Name

XieTecGeomGaussian - allocate and fill an `XieGeomGaussianParam` structure.

Synopsis

```
XieGeomGaussianParam *XieTecGeomGaussian (sigma, normalize, radius,
            simple)
   double sigma;
   double normalize;
   unsigned int radius;
   Bool simple;
```

Arguments

sigma	Specifies the drop-off rate.
normalize	Specifies a normalization constant.
radius	Specifies the extent of computation.
simple	Specifies the approximation form to use.

Returns

The `XieGeomGaussianParam` structure.

Description

`XieGeomGaussian()` allocates and returns a pointer to an `XieGeomGaussianParam` structure. The returned structure represents the list of parameters required by the geometry technique and may be used as the *sample_param* argument of `XieFloGeometry()` (when the *sample_tech* argument is `xieValGeomGaussian`).

If insufficient memory is available, `XieTecGeomGaussian()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

A Geometry element can be visualized as stepping through each possible output pixel location in turn, and computing the location from which to obtain each input pixel value. Each pixel (x',y') in the output image is mapped to the coordinate location (x,y) in *src* by:

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{pmatrix} \begin{pmatrix} \mathbf{x'} \\ \mathbf{y'} \end{pmatrix} + \begin{pmatrix} \mathbf{t_x} \\ \mathbf{t_y} \end{pmatrix}$$

It is not unusual that the input location (x,y) so derived will be non-integral, and will not correspond exactly to a single pixel in the input image.

From sampling theory, a bandwidth limited continuous input image can be recovered perfectly (under certain conditions) from its sampled output by computing the convolution:

$$I(x,y) = \sum_m \sum_n i(m,n)h(x-m,y-n)$$

Here $I(x,y)$ is the continuous image, $i(m,n)$ the discrete sampled image, and $h(u,v)$ is the impulse response function for an appropriate low pass filter. The specific form of h(u,v) for a Gaussian impulse response function is given by:

$$h(u,v) = \frac{1}{2\pi\sigma^2} \times \exp\left(-\frac{u^2+v^2}{2\sigma^2}\right)$$

The $1/2\pi\sigma^2$ term is called the "normalization constant" and may be altered using the *`normalize`* parameter. The suggested value for σ (*`sigma`*, the drop-off rate) is 1. Note that all technique parameters must be chosen in concert

`radius` defines the extent of computation. A suggested value for *`radius`* is one, that is, only pixels within a distance of one in either the *`x`* or *`y`* direction are involved in the calculation.

For computational convenience, a simplified form of Gaussian interpolation is provided. If *`simple`* is `True`, the impulse-response function h(u,v) is:

$$h(u,v) = N \times 2^{-\frac{u^2+v^2}{\sigma^2}}$$

The normalization factor *N* is defined by *`normalize`*. As with true Gaussian interpolation, the *`radius`* parameter is used to determine the number of pixels involved in the computation.

Structures

`XieGeomGaussian()` sets the structure field `sigma` to the value of the argument *`sigma`*; the structure field `normalize` to the value of the argument *`normalize`*; the structure field `radius` to the value of the argument *`radius`*; and the structure field `simple` to the value of the argument *`simple`*.

```
typedef struct {
   float sigma;
   float normalize;
   unsigned int radius;
   Bool simple;
} XieGeomGaussianParam;
```

See Also

XieFloGeometry().

Name

XieTecGeomNearestNeighbor - allocate and fill an `XieGeomNearestNeighborParam` structure.

Synopsis

```
XieGeomNearestNeighborParam *XieTecGeomNearestNeighbor (modify)
       unsigned int modify;
```

Arguments

modify	Specifies technique behavior on even boundaries.

Returns

The `XieGeomNearestNeighborParam` structure.

Description

`XieGeomNearestNeighbor()` allocates and returns a pointer to an `XieGeomNearestNeighborParam` structure. The returned structure represents the list of parameters required by the geometry technique and may be used as the *sample_param* argument of `XieFloGeometry()` (when the *sample_tech* argument is `xieValGeomNearestNeighbor`).

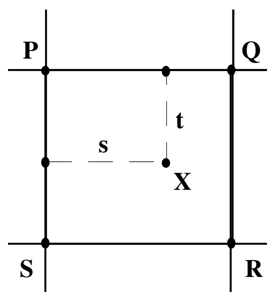
If insufficient memory is available, `XieTecGeomNearestNeighbor()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

A Geometry element can be visualized as stepping through each possible output pixel location in turn, and computing the location from which to obtain each input pixel value. Each pixel (x',y') in the output image is mapped to the coordinate location (x,y) in *src* by:

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{pmatrix} \begin{pmatrix} \mathbf{x'} \\ \mathbf{y'} \end{pmatrix} + \begin{pmatrix} \mathbf{t_x} \\ \mathbf{t_y} \end{pmatrix}$$

It is not unusual that the input location (x,y) so derived will be non-integral, and will not correspond exactly to a single pixel in the input image.

To illustrate NearestNeighbor technique, assume that the pixel grid locations P, Q, R, and S are integral. Pixel location $X = (x,y)^T$, obtained from the mapping equation above, differs from P by fractional amounts s in the *x* direction and t in the *y* direction.



Let I(P) be the value of the input image at coordinate P, if P is within the image extent. Otherwise, let I(P) be *constant*, where *constant* is the pixel value passed to the Geometry element. A value of I(X) must be estimated from I(P), I(Q), I(R), and I(S). In nearest-neighbor sampling, we simply choose the image value from the discrete location closest to X. Thus,

if s < 1/2, t < 1/2, set I(X) = I(P),
if s > 1/2, t < 1/2, set I(X) = I(Q),
if s > 1/2, t > 1/2, set I(X) = I(R),
if s < 1/2, t > 1/2, set I(X) = I(S).

The behavior on even boundaries (s = *1/2* or t = *1/2*) is determined by the *modify* parameter. One of the standard nearest neighbor modify values can be assigned to *modify*:

```
xieValFavorDown
xieValFavorUp
xieValRoundNW
xieValRoundNE
xieValRoundSE
xieValRoundSW
```

If *modify* is `xieValFavorDown`, all less than signs in the above are replaced with less than or equal signs. Thus, P would win all ties, S and Q would lose to P but win over R, and R would lose all ties. If *modify* is `xieValFavorUp`, then all greater than signs would be replaced with greater than or equals, and the opposite behavior would occur. Four additional options are provided. The `xieValRoundxx` options will always choose a specific integral pixel grid location; these are not strictly nearest neighbor algorithms but are available for computational/filtering convenience.

Structures

`XieGeomNearestNeighbor()` sets the structure field `modify` to the value of the argument *modify*.

```
typedef struct {
   unsigned int modify;
} XieGeomNearestNeighborParam;
/* Definitions of NearestNeighbor Modify */
#define xieValFavorDown               1
#define xieValFavorUp                 2
#define xieValRoundNW                 3
#define xieValRoundNE                 4
#define xieValRoundSE                 5
#define xieValRoundSW                 6
```

See Also

XieFloGeometry().

Name

XieTecHistogramGaussian - allocate and fill an `XieHistogramGaussianParam` structure.

Synopsis

```
XieHistogramGaussianParam *XieTecHistogramGaussian (mean, sigma)
   double mean;
   double sigma;
```

Arguments

mean	Specifies the center of the Gaussian probability density function.
sigma	Specifies the "spread" of the Gaussian probability density function.

Returns

The `XieHistogramGaussianParam` structure.

Description

`XieHistogramGaussian()` allocates and returns a pointer to an `XieHistogramGaussianParam` structure. The returned structure represents the list of parameters required by the match-histogram shape technique and may be used as the *shape_param* argument of `XieFloMatchHistogram()` (when the *shape* argument is `xieValHistogramGaussian`).

If insufficient memory is available, `XieTecHistogramGaussian()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The Gaussian match-histogram shape technique specifies that the output image is to have a histogram that approximates a Gaussian probability density. The supplied parameters are used to generate a Gaussian probability density function centered around the *mean* level with a “spread” specified by *sigma*:

$$\boldsymbol{P}_{Gaussian}(x) = \frac{1}{\sqrt{2\pi \times sigma^2}} \times \exp\left(\frac{-(x - mean)^2}{2 \times sigma^2}\right)$$

Structures

`XieHistogramGaussian()` sets the structure field `mean` to the value of the argument *mean*; and the structure field `sigma` to the value of the argument *sigma*.

```
typedef struct {
   float mean;
   float sigma;
} XieHistogramGaussianParam;
```

See Also

XieFloMatchHistogram().

Name

XieTecHistogramHyperbolic - allocate and fill an `XieHistogramHyperbolicParam` structure.

Synopsis

```
XieHistogramHyperbolicParam *XieTecHistogramHyperbolic (constant,
            shape_factor)
   double constant;
   Bool shape_factor;
```

Arguments

constant	Specifies a value used to generate a hyperbolic probability density function
shape_factor	Specifies the relationship between the histogram shape and image levels.

Returns

The `XieHistogramHyperbolicParam` structure.

Description

`XieHistogramHyperbolic()` allocates and returns a pointer to an `XieHistogramHyperbolicParam` structure. The returned structure represents the list of parameters required by the match-histogram shape technique and may be used as the *shape_param* argument of `XieFloMatchHistogram()` (when the *shape* argument is `xieValHistogramHyperbolic`).

If insufficient memory is available, `XieTecHistogramHyperbolic()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

The hyperbolic match-histogram shape technique specifies that the output image is to have a histogram that approximates a hyperbolic probability density.

constant is used to generate a hyperbolic probability density function:

$$\boldsymbol{P}_{Hyperbolic}(x) = \frac{1}{(x + \mathtt{constant}) \times \log\left(1 + \dfrac{1}{\mathtt{constant}}\right)}$$

shape_factor should be specified as `False` if the histogram shape represents decreasing values for higher *levels*, or `True` if the shape represents increasing values for higher *levels*.

Structures

`XieHistogramHyperbolic()` sets the structure field `constant` to the value of the argument *constant*; and the structure field `shape_factor` to the value of the argument *shape_factor*.

```
typedef struct {
   float constant;
   Bool shape_factor;
} XieHistogramHyperbolicParam;
```

See Also

XieFloMatchHistogram().

Name

XieTecWhiteAdjustCIELabShift - allocate and fill an `XieWhiteAdjustCIELabShiftParam` structure.

Synopsis

```
XieWhiteAdjustCIELabShiftParam *XieTecWhiteAdjustCIELabShift
            (white_point)
   XieConstant white_point;
```

Arguments

white_point Specifies the white point of the (source or output) data.

Returns

The `XieWhiteAdjustCIELabShiftParam` structure.

Description

`XieWhiteAdjustCIELabShift()` allocates and returns a pointer to an `XieWhiteAdjustCIELabShiftParam` structure. The returned structure represents the list of parameters required by the WhiteAdjust technique and may be used as the *white_adjust_param* argument of `XieTecRGBToCIELab()`, `XieTecRGBToCIEXYZ()`, `XieTecCIELabToRGB()`, and `XieTecCIEXYZToRGB()` (when the *white_adjust_tech* argument is `xieValWhiteAdjustCIELabShift`).

If insufficient memory is available, `XieTecWhiteAdjustCIELabShift()` returns `NULL`. To free the memory allocated to this structure, use `XFree()`.

White point correction can be used to ensure that white "looks" white, or it can be used to change the overall tint of an image.

The CIELabShift WhiteAdjust technique specifies that white point correction is to be accomplished by adding the white point displacement to the ***ab*** plane in the CIELab colorspace. The *white_point* is specified using CIEXYZ encodings. If the WhiteAdjust technique is used with a color conversion technique that converts from RGB, *white_point* specifies the desired white point of the output data; if the conversion is to RGB, *white_point* specifies the white point of the source data.

Structures

`XieWhiteAdjustCIELabShift()` sets the structure field `white_point` to the value of the argument *white_point*.

```
typedef float XieConstant[3];
typedef unsigned XieWhiteAdjustTechnique;
typedef struct {
   XieConstant white_point;
} XieWhiteAdjustCIELabShiftParam;
```

See Also

XieTecRGBToCIELab(), XieTecRGBToCIEXYZ(), XieTecCIELabToRGB(), XieTecCIEXYZToRGB().

Name

XieFreeTechniques - free memory allocated for a list of techniques.

Synopsis

```
void XieFreeTechniques (techs, count)
   XieTechnique *techs;
   unsigned int count;
```

Arguments

techs	Specifies the list of techniques to be freed.
count	Specifies the number of items in the list of techniques to be freed.

Description

`XieFreeTechniques()` frees the memory previously allocated for *techs*. Care should be taken that the argument pair *techs/count* match an argument pair *techniques_ret/ ntechniques_ret* returned from `XieQueryTechniques()`.

See `XieQueryTechniques()` for a description of the `XieTechnique` structure.

Structures

```
typedef unsigned XieTechniqueGroup;
typedef struct {
    Bool needs_param;
    XieTechniqueGroup group;
    unsigned int number;
    unsigned int speed;
    char *name;
} XieTechnique;
```

See Also

XieQueryTechniques().

Name

XieFreePhotofloGraph - free memory allocated for an array of `XiePhotoElement` structures.

Synopsis

```
void XieFreePhotofloGraph (elements, count)
   XiePhotoElement *elements;
   unsigned int count;
```

Arguments

elements	Specifies the array of `XiePhotoElement` structures to be freed.
count	Specifies the number of `XiePhotoElement` structures in the array.

Description

`XieFreePhotofloGraph()` frees the specified array of `XiePhotoElement` structures.

Care should be taken that the argument pair *elements/count* match a returned value (an array of `XiePhotoElement` structures) and argument *count* from a call to `XieAllocatePhotofloGraph()`.

Technique parameters are not freed by using `XieFreePhotofloGraph()`. Most of the technique parameters, with the exception of the JPEG baseline and JPEG lossless encode techniques, should be freed using `XFree()`. This is so the client can reuse technique parameters between photoflos.

Structures

```
typedef struct {
   int elemType;
   /* union of ALL element types */
   union {
      ...
      ...
   } data;
} XiePhotoElement;
```

See Also

XieAllocatePhotofloGraph(), XieCreatePhotoflo(), XieModifyPhotoflo(), XieRedefinePhotoflo(), XieExecutePhotoflo(), XieQueryPhotoflo(), XieDestroyPhotoflo(), XieExecuteImmediate().

Name

XieFreeEncodeJPEGBaseline - free the memory allocated to the structure `XieEncodeJPEGBaselineParam`.

Synopsis

```
void XieFreeEncodeJPEGBaseline (param)
   XieEncodeJPEGBaselineParam *param
```

Arguments

param	Specifies a pointer to the structure that is to be freed.

Description

`XieFreeEncodeJPEGBaseline()` (rather than `XFree()`) should be used to free the memory allocated by `XieEncodeJPEGBaseline()`.

Structures

```
typedef struct {
   XieInterleave interleave;
   XieOrientation band_order;
   unsigned char horizontal_samples[3];
   unsigned char vertical_samples[3];
   char *q_table;
   unsigned int q_size;
   char *ac_table;
   unsigned int ac_size;
   char *dc_table;
   unsigned int dc_size;
} XieEncodeJPEGBaselineParam;
```

See Also

XieTecEncodeJPEGBaseline().

Name

XieFreeEncodeJPEGLossless - free the memory allocated to the structure `XieEncodeJPEGLosslessParam`.

Synopsis

```
void XieFreeEncodeJPEGLossless (param)
   XieEncodeJPEGLosslessParam *param;
```

Arguments

param	Specifies a pointer to the structure that is to be freed.

Description

`XieFreeEncodeJPEGLossless()` (rather than `XFree()`) should be used to free the memory allocated by `XieEncodeJPEGLossless()`.

Note that the JPEG Lossless technique is not available in the R6 sample implementation of XIE.

Structures

```
typedef struct {
   XieInterleave interleave;
   XieOrientation band_order;
   unsigned char predictor[3];
   char *table;
   unsigned int table_size;
} XieEncodeJPEGLosslessParam;
```

See Also

XieTecEncodeJPEGLossless().

Name

XieFreePasteUpTiles - free the memory allocated to the tiles field of a PasteUp structure.

Synopsis

```
void XieFreePasteUpTiles (element)
   XiePhotoElement *element;
```

Arguments

element	Specifies the `XiePhotoElement` structure to use.

Description

`XieFreePasteUpTiles()` frees the memory allocated to the `tiles` field in the specified `PasteUp` member structure; after the memory has been freed, the field value is set to `NULL`.

Structures

```
typedef struct {
   int elemType;
   union {
      ...
      struct {
         unsigned int width;
         unsigned int height;
         XieConstant constant;
         XieTile *tiles;
         unsigned int tile_count;
      } PasteUp;
      ...
   } data;
} XiePhotoElement;
```

See Also

XieFloPasteUp().

Description

The client is notified that a ConvertToIndex element has completed color allocation, but has produced a result of lesser fidelity than was requested using the technique that was specified for the ConvertToIndex element.

The structure fields `name_space`, `flo_id`, `src`, and `elem_type` identify the photoflo and specific ConvertToIndex element from which the `ColorAlloc` event originated. The structure field `time` is the server time when the `ColorAlloc` event occurred, in milliseconds. The structure field `color_list` is the color list resource that received the allocated colors. The structure field `color_alloc_technique` is the ColorAlloc technique specified to the ConvertToIndex element. The structure field `color_alloc_data` can be used for other information which is specific to the ColorAlloc technique.

Structures

```
/* ColorAlloc Event Code */
#define xieEvnNoColorAlloc             0

typedef struct {
   int type;
   unsigned long serial;
   Bool send_event;
   Display *display;
   unsigned long name_space;
   Time time;
   unsigned long flo_id;
   XiePhototag src;
   unsigned int elem_type;
   XieColorList color_list;
   XieColorAllocTechnique color_alloc_technique;
   unsigned long color_alloc_data;
} XieColorAllocEvent;
```

See Also

XieFloConvertToIndex(), XieTecColorAllocAll().

Description

A `DecodeNotify` event notifies the client that anomalies were encountered while decoding a compressed image (see the *notify* arguments of `XieImportClientPhoto()` and `XieImportPhotomap()`). Either an error has been encountered while decoding an image, or the image data received does not satisfy the expected dimensions.

The structure fields `name_space`, `flo_id`, `src`, and `elem_type` identify the photoflo and element from which the `DecodeNotify` event originated. The structure field `time` is the server time when the `DecodeNotify` event occurred, in milliseconds. The structure field `band_number` associates the event with a specific band of the image. The structure field `decode_technique` is the Decode technique being used. The structure fields `width` and `height` are the dimensions discovered while decoding the data. The structure field `aborted` is `True` if decoding was aborted, or `False` if recovery was possible.

Recovery from a decode error may result in some missing or garbled image data. This may also cause the height of the decoded data to be less than was expected. If the structure fields `width` or `height` do not match the width and height specified to `XieImportClientPhoto()`, the image data is clipped or padded (with zeros), as required, to enforce the `XieImportClientPhoto()` specified dimensions.

Structures

```
/* DecodeNotify Event Code */
#define xieEvnNoDecodeNotify           1

typedef struct {
   int type;
   unsigned long serial;
   Bool send_event;
   Display *display;
   unsigned long name_space;
   Time time;
   unsigned long flo_id;
   XiePhototag src;
   unsigned int elem_type;
   XieDecodeTechnique decode_technique;
   Bool aborted;
   unsigned int band_number;
   unsigned long width;
   unsigned long height;
} XieDecodeNotifyEvent;
```

See Also

XieFloImportClientPhoto(), XieFloImportPhotomap().

Description

The client is notified that an ExportClient element has data available (see the *notify* argument of the applicable XieExportClient...() function). If *notify* was specified as xieValFirstData, this event will only be sent the first time data become available from the ExportClient element. Otherwise (i.e. xieValNewData was specified) this event will be generated each time the amount of data available changes from zero to non-zero.

The structure fields name_space, flo_id, src, and elem_type identify the photoflo and specific ExportClient element from which the ExportAvailable event originated. The structure field time is the server time when the ExportClient event occurred, in milliseconds. The structure field band_number associates the event with a specific band of the image or LUT. The structure field data is information specific to elem_type (e.g. the number of LUT entries or ROI rectangles available).

Where there is a single ExportClient element, the client can just read bytes or be event-driven. For photoflos containing multiple ExportClient elements, the client should be event-driven.

Structures

```
/* ExportAvailable Event Code */
#define xieEvnNoExportAvailable        2

typedef struct {
   int type;
   unsigned long serial;
   Bool send_event;
   Display *display;
   unsigned long name_space;
   Time time;
   unsigned long flo_id;
   XiePhototag src;
   unsigned int elem_type;
   unsigned int band_number;
   unsigned long data[3];
} XieExportAvailableEvent;
```

See Also

XieFloExportClientHistogram(), XieFloExportClientLUT(), XieFloExportClientPhoto(), XieFloExportClientROI().

Description

The client is notified when an ImportDrawable or ImportDrawablePlane element encounters obscured regions in a Window that cannot be retrieved from backing store (see the `notify` argument of the import element routine). A separate `ImportObscured` event is returned for each affected region.

The structure fields `name_space`, `flo_id`, and `src` identify the photoflo and the specific import element from which the `ImportObscured` event originated. The structure field `time` is the server time when the `ImportObscured` event occurred, in milliseconds. The structure field `window` identifies the Window. The obscured region of the window is specified by the structure fields `x`, `y`, `width`, and `height`.

Note: image data within obscured regions will be populated with the `fill` argument supplied to the import element.

Structures

```
/* ImportObscured Event Code */
#define xieEvnNoImportObscured         3

typedef struct {
   int type;
   unsigned long serial;
   Bool send_event;
   Display *display;
   unsigned long name_space;
   Time time;
   unsigned long flo_id;
   XiePhototag src;
   unsigned int elem_type;
   Window window;
   int x;
   int y;
   unsigned int width;
   unsigned int height;
} XieImportObscuredEvent;
```

See Also

XieFloImportDrawable(), XieFloImportDrawablePlane().

Description

A PhotofloDone event notifies the client that a photoflo has left the active state. It is enabled by the *notify* argument of XieExecutePhotoflo() or XieExecuteImmediate().

The photoflo from which the PhotofloDone event originated is identified by the structure fields name_space and flo_id. The structure field time is the server time when the PhotofloDone event occurred, in milliseconds. The reason the photoflo left the active state is indicated by the structure field type.

If the Photoflo terminated due to an error condition, the details concerning the error have preceded this event in an error message.

Structures

```
/* PhotofloDone Event Code */
#define xieEvnNoPhotofloDone           4

typedef struct {
   int type;
   unsigned long serial;
   Bool send_event;
   Display *display;
   unsigned long name_space;
   Time time;
   unsigned long flo_id;
} XiePhotofloDoneEvent;
```

See Also

XieExecuteImmediate(), XieExecutePhotoflo().

Resource Errors

The following error codes are allocated from the extension error space to provide for the errors returned by XIE:

Error	Cause
`xieErrNoColorlist`	the value for a *color_list* argument does not name a defined color list.
`xieErrNoLUT`	the value for a *lut* argument does not name a defined LUT.
`xieErrNoPhotoflo`	the value for a *photoflo* argument does not name a defined photoflo.
`xieErrNoPhotomap`	the value for a *photomap* argument does not name a defined photomap.
`xieErrNoPhotospace`	the value for a *photospace* argument does not name a defined photospace.
`xieErrNoROI`	the value for a *roi* argument does not name a defined ROI.
`xieErrNoFlo`	an error has been detected while defining, executing, or accessing a photoflo (See Photoflo Errors).

XIE also uses the core protocol BadAccess, BadAlloc, BadIDChoice, BadLength, BadRequest, and BadValue errors.

Photoflo Errors

If an error is detected while defining, executing, or accessing a photoflo, an `xieErrNoFlo...` error is returned. This single error code is allocated from the extension error space for all photoflo related errors. The following sub-codes are defined to provide the details of the error:

Error	Cause
`xieErrNoFloAccess`	attempt to execute, modify, or redefine an active photoflo *or* attempt to Get/Put client data from/to an inactive photoflo.
`xieErrNoFloAlloc`	insufficient resources (e.g. memory).
`xieErrNoFloColormap`	an unknown Colormap has been specified.
`xieErrNoFloColorList`	an unknown color list has been specified.
`xieErrNoFloDomain`	invalid domain phototag: - source data is not a list-of-rectangles or control-plane *or* - specified non-zero on a DIS server.
`xieErrNoFloDrawable`	an unknown `Drawable` has been specified.
`xieErrNoFloElement`	an unknown element type has been specified *or* invalid element type for request *or* attempt to change or add an element type.
`xieErrNoFloGC`	an unknown GContext has been specified.
`xieErrNoFloID`	invalid executable: - an unknown photoflo has been specified *or* - an unknown photospace has been specified.
`xieErrNoFloLength`	an element was received with the incorrect number of bytes.
`xieErrNoFloLUT`	an unknown LUT has been specified.
`xieErrNoFloMatch`	some argument or pair of arguments has the correct type and range, but it fails to match in some other way required by the element.
`xieErrNoFloOperator`	an unknown operator has been specified.
`xieErrNoFloPhotomap`	an unknown photomap has been specified.
`xieErrNoFloROI`	an unknown ROI has been specified.
`xieErrNoFloSource`	an invalid phototag has been specified: - zero, but a phototag is required *or* - downstream from the particular element *or* - beyond the bounds of the photoflo.
`xieErrNoFloTechnique`	an unknown technique has been specified *or* invalid technique specific-parameters have been specified *or* the wrong number of technique-specific parameters have been given.
`xieErrNoFloValue`	some numeric value falls outside of the range of values accepted by the element.
`xieErrNoFloImplementation`	some aspect of a request is not implemented by the server: the client should be prepared to receive and handle this error.

Structures

```
/* Definition of Error Codes */
#define xieErrNoColorList             0
#define xieErrNoLUT                   1
#define xieErrNoPhotoflo              2
#define xieErrNoPhotomap              3
#define xieErrNoPhotospace            4
#define xieErrNoROI                   5
#define xieErrNoFlo                   6

/* Definitions of Flo Error (Sub-) Codes */
#define xieErrNoFloAccess             1
#define xieErrNoFloAlloc              2
#define xieErrNoFloColormap           3
#define xieErrNoFloColorList          4
#define xieErrNoFloDomain             5
#define xieErrNoFloDrawable           6
#define xieErrNoFloElement            7
#define xieErrNoFloGC                 8
#define xieErrNoFloID                 9
#define xieErrNoFloLength             10
#define xieErrNoFloLUT                11
#define xieErrNoFloMatch              12
#define xieErrNoFloOperator           13
#define xieErrNoFloPhotomap           14
#define xieErrNoFloROI                15
#define xieErrNoFloSource             16
#define xieErrNoFloTechnique          17
#define xieErrNoFloValue              18
#define xieErrNoFloImplementation     19
```