4

Resources

Overview

XIE resources are extension server objects created by the client which contain information used by the extension in carrying out various protocol requests. There are protocol requests to create and destroy each resource type. Resource identifiers are generated by the client as defined by the core protocol. All XIE resources are reference counted.

Binding Resources to Photoflos

Each resource that is referenced from a **Photoflo** is bound to the **Photoflo** (and its reference count is incremented) when the **Photoflo** becomes ***Active***.

ColorList resources

A **ColorList** is purged of previous allocations when a **Photoflo** begins execution, and is populated with new allocations (via **ConvertToIndex**) as the **Photoflo** proceeds. A **ColorList** can only be referenced by one ***Active*** **Photoflo** at a time. The **ColorList**'s reference count is decremented when the **Photoflo** is no longer ***Active***.

LUT, Photomap, and ROI resources

These XIE resources consist of two parts: one part contains the resource-id, the other part holds attributes and data. Figure 4-1 pictures the first part as a *handle*, and the second part as a *bucket*. Upon resource creation only the handle exists. The bucket is created when the **Photoflo** (referencing the handle via an `ExportResource` element) becomes ***Active***. The handle and bucket are joined when the **Photoflo** successfully completes. Conceptually each part contains a separate reference count.

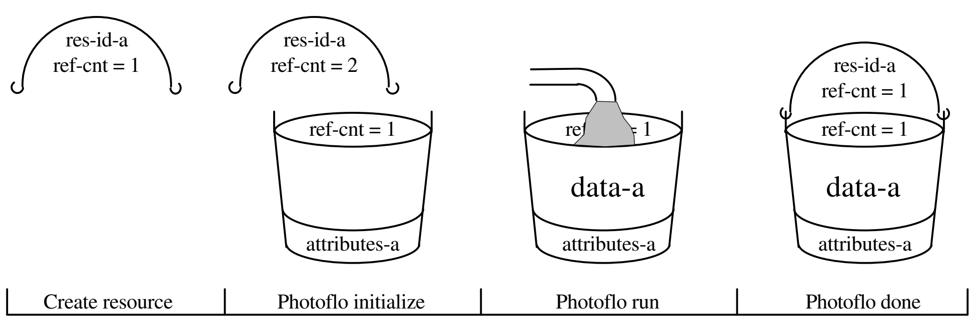

Figure 4-1 Creating and populating a new Photomap

In the case of `ImportResource` elements the handle and bucket are both bound to the **Photoflo** when it first becomes ***Active*** (initialization). For `ExportResource` elements, only the handle is bound at this time and a new bucket is created to hold the forthcoming data. This bucket's attributes are derived from the `ExportResource` element.

If an export handle is already connected to a bucket, the old bucket remains with its handle until successful **Photoflo** completion. Thus, the old bucket is preserved if the **Photoflo** terminates due to an error condition or if an **Abort** request is received from the client.

The new bucket is filled with data as the **Photoflo** proceeds. When the **Photoflo** successfully completes, old buckets can be freed if they have no other references, and the waiting handle is joined to the new bucket. This is illustrated in figure 4-2.

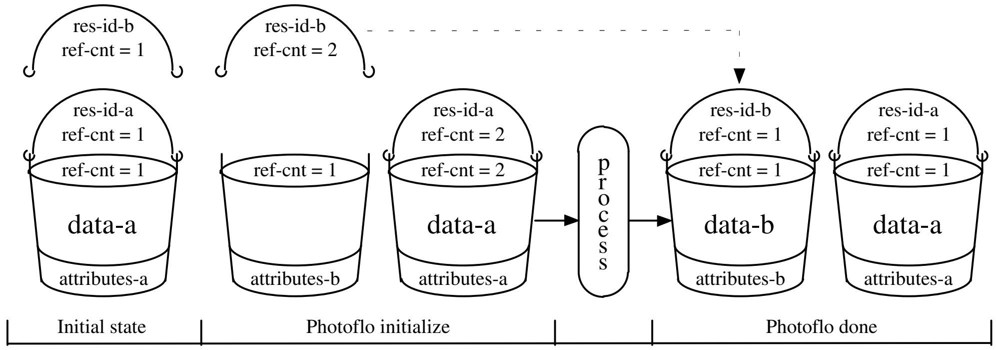

Figure 4-2 Process image from Photomap *a*, place result into Photomap *b*

This model also allows pseudo in-place operations such as is illustrated in figure 4-3. Here an image is imported from a **Photomap**, processed, and then the result is exported back to the same **Photomap**. Note that upon completion the old handle is moved to the new bucket and the old bucket is destroyed.

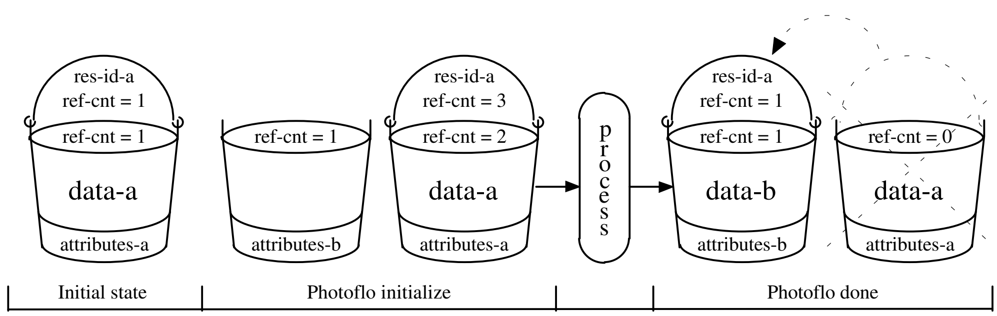

Figure 4-3 Process image from Photomap *a* "in-place"

Resource destruction

When a *destroy* request is received the resource-id is immediately disassociated from the resource and the resource's reference count(s) is(are) decremented. For each reference count that goes to zero, the associated resources are freed (e.g. memory or colors).

Synchronizing resource access

If multiple **Photoflo**s reference the same resource, the client must synchronize access to the resource (e.g. **Await** completion of the **Photoflo** writing the resource before beginning execution of a **Photoflo** reading from the resource). Core resources can be volatile, requiring the client to maintain the data integrity of the resource while the **Photoflo** is active.

Capability Acquisition

QueryImageExtension

XieReqQueryImageExtension

client-major-version	`CARD16`
client-minor-version	`CARD16`
→	
server-major-version	`CARD16`
server-minor-version	`CARD16`
service-class	XieTypServiceClass
alignment	XieTypAlignment
unconstrained-mantissa	`CARD16`
unconstrained-max-exp	`INT32`
unconstrained-min-exp	`INT32`
constrained-levels	`LISTofCARD32`

Errors: ***Alloc***
Events: none

QueryImageExtension returns information about the XIE server's capabilities. Each client should use **QueryImageExtension** to establish version compatibility between client and server prior to making any other XIE request. If a client fails to first establish the desired version using **QueryImageExtension**, the behavior of other requests is "undefined" (which generally means that the server will use the version number of its own choice).

Client-major-version and *client-minor-version* specify which version of the XIE protocol the client would like to use. If the client can support multiple versions, the highest version should be given.

The *server-major-version* and *server-minor-version* specify the version of the XIE protocol that the server expects from the client. If the server supports the version requested by the client, this version number will be returned. If the client has requested a higher version than is supported by the server, the server's highest version will be returned. Otherwise, if the client has requested a lower version than is supported by the server, the server's lowest version will be returned. It is the client's responsibility to decide whether or not it can match the server's version of the protocol.

Service-class specifies the **ServiceClass** supported by the XIE server[1]. *Alignment* specifies the image data alignment restrictions of the server (i.e. the alignment of pixels and scanlines).

The following parameters convey the approximate range and granularity of ***Unconstrained*** data in the XIE server. For servers that represent ***Unconstrained*** data using floating point, *unconstrained-mantissa* returns the number of bits in the server's floating point format (including the sign bit). If the server uses fixed point, *unconstrained-mantissa* is 0. *Unconstrained-max-exp* returns the largest value n such that $2^n - 1$ is representable in the server's ***Unconstrained*** data format. *Unconstrained-min-exp* returns the smallest (most negative) value n such that 2^n is representable in the server's ***Unconstrained*** data format.

Constrained-levels provides a hint about how the XIE server might process ***Constrained*** data most efficiently. *Constrained-levels* returns a list of *levels* that are "recommended" for ***Constrained*** data by the server. (a value of 0 means 2^{32} *levels*).

Error	Cause
Alloc	insufficient resources

[1] ***ServiceClass***es defined in this document include: ***Full*** XIE and ***DIS***.

Technique Acquisition

QueryTechniques

XieReqQueryTechniques

technique-group	XieTypTechniqueGroup
→	
technique-information	*ListofXieTypTechniqueRec*

Errors: ***Alloc, Value***
Events: none

QueryTechniques returns information about the standard and private techniques that are supported by the server. The server may be queried for ***All*** techniques, all ***Default*** techniques, or a group of techniques that are functionally similar (e.g. all ***Geometry*** techniques).

Technique-group specifies the group of techniques for which the server is to return information.

Technique-information is a list of **TechniqueRec** entries which are returned in arbitrary order. Each entry contains the following information:

needs-parameters	if true, indicates that the technique requires additional parameters; if false, 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.
group	identifies which group the technique belongs to.
number	the numeric identifier assigned to the technique (MS bit is zero for *standard* techniques, or one for *private* techniques).
speed	the server's assessment of the speed of this technique relative to other techniques in the same group (0 :== slowest, 255 :== fastest).
name	the XIE compliant technique name string of the form: <STANDARD-TECHNIQUE-NAME> or _<ORGANIZATION-NAME>_<PRIVATE-TECHNIQUE-NAME>

The technique *number* is supplied to pipeline elements to specify a desired algorithm or technique. While *numbers* for standard techniques can be hard-coded (e.g. defined in an include file), *numbers* for private techniques must be obtained using the **QueryTechniques** protocol request prior to their use.

Error	**Cause**
Alloc	insufficient resources
Value	unknown *technique-group*

A complete description of each technique and its parameters is given in Appendix A. A summary of standard techniques itemized by service class can be found in Appendix B. Numbers assigned to standard techniques are encoded in Appendix C.

ColorList

CreateColorList

XieReqCreateColorList

color-list	XieTypColorList

Errors: ***Alloc, IDChoice***
Events: none

CreateColorList creates an unpopulated server resource which can be used to store the list of colors allocated by a **ConvertToIndex** element. The COLORMAP allocations that are recorded in a **ColorList** belong to the client that executed the **Photoflo** that populated the resource (this is not necessarily the same client that created the **ColorList**). A **ColorList** cannot be the target of more than one ***Active*** **Photoflo** at a time.

Color-list is the client supplied **ColorList** identifier to be assigned to this resource.

Color-list is populated (or re-populated) with new COLORMAP allocations via a **ConvertToIndex** element as the **Photoflo** executes. The contents of *color-list* may be queried using **QueryColorList**.

The client may explicitly purge all allocated cells from *color-list* using **PurgeColorList**. The client may cause an implicit deallocation of cells from *color-list* by making it the target of a **Photoflo** and commencing its execution. An implicit purge also takes place if the COLORMAP referenced by *color-list* is freed or if the client that owns the cells exits (i.e. the client that populated *color-list*).

Color-list can be destroyed using **DestroyColorList**.

Error	Cause
Alloc	insufficient resources
IDChoice	invalid *color-list*

DestroyColorList

XieReqDestroyColorList

color-list	XieTypColorList

Errors: ***ColorList***
Events: none

DestroyColorList disassociates the resource-id from *color-list* and decrements its reference count. If there are no other references, it frees colors held in *color-list*, and then destroys the **ColorList** identified by *color-list*.

Error	Cause
ColorList	invalid *color-list*

PurgeColorList

XieReqPurgeColorList

color-list	XieTypColorList

Errors: ***Access, ColorList***
Events: none

PurgeColorList frees the colors from the **ColorList** identified by *color-list*.

Error	Cause
Access	attempt to purge colors when *color-list* is being written by a **Photoflo**
ColorList	invalid *color-list*

QueryColorList

XieReqQueryColorList

color-list	XieTypColorList
→	
colormap	`COLORMAP`
colors	`LISTofCARD32`

Errors: ***Alloc, ColorList***
Events: none

QueryColorList returns a list of colors allocated by a **ConvertToIndex** element.

Color-list is the **ColorList** resource to be queried.

Colormap is the `COLORMAP` from which the colors were allocated. *Colors* is the list of allocated `COLORMAP` indices.

When there are no colors in *color-list*, the value zero (0) is returned for *colormap*, and the list of *colors* is of length zero.

Error	Cause
Alloc	insufficient resources
ColorList	invalid *color-list*

Notes:
- The `COLORMAP` was originally supplied by the client as a **ConvertToIndex** parameter.
- The returned colors are owned by the server, and therefore, should not be freed via core X protocol requests (e.g. FreeColors)
- The allocated colors can be freed by:
  – issuing a PurgeColorList or DestroyColorList request,
  – commencing execution of a Photoflo that targets *color-list*,
  – freeing *colormap*,
  – shutting down the client that populated *color-list*.

LUT

CreateLUT

XieReqCreateLUT

lut	XieTypLUT

Errors: ***Alloc, IDChoice***
Events: none

CreateLUT creates a server resource which is used as a lookup table by the **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.

Lut is the client supplied **LUT** identifier to be assigned to this resource.

The **LUT** is populated (or re-populated) with lookup table entries after the successful execution of a **Photoflo** containing an **ExportLUT** element which targets *lut*. Lookup table data can be imported into a **Photoflo** using an **ImportLUT** element. These data can be used by **Point**, and they can be exported to the client with the aid of an **ExportClientLUT** element.

Lut can be destroyed using **DestroyLUT**.

Error	Cause
Alloc	insufficient resources
IDChoice	invalid *lut*

A **LUT** can be populated with a simple **ExecuteImmediate**(**ImportClientLUT**, **ExportLUT**) **Photoflo**. **PutClientData** is then used to transport the lookup table entries. See **ImportClientLUT** and **Point** for further information on **LUTs**

DestroyLUT

XieReqDestroyLUT

lut	XieTypLUT

Errors: ***LUT***
Events: none

DestroyLUT disassociates the resource-id from *lut* and decrements its reference count. If there are no other references, it destroys the **LUT** identified by *lut*.

Error	Cause
LUT	invalid *lut*

Photomap

CreatePhotomap

XieReqCreatePhotomap

photomap	XiePhotomap

Errors: ***Alloc, IDChoice***
Events: none

Attribute	Value
class	undefined (see text)
type	undefined (see text)
width	undefined (see text)
height	undefined (see text)
levels	undefined (see text)

CreatePhotomap creates a server resource which is used to store image data. **Photomap** data may be rendered for display or used as input to control or modify the rendition of another image.

Photomap is the client supplied **Photomap** identifier to be assigned to this resource.

Photomap attributes are defined when a **Photoflo** containing an **ExportPhotomap** element populates *photomap* with data.

The **Photomap** is populated (or re-populated) with image data after the successful execution of a **Photoflo** containing an **ExportPhotomap** element that targets *photomap*. *Photomap* data can be imported into a **Photoflo** for rendition or control purposes using an **ImportPhotomap** element. It can also be exported back to the client with the aid of an **ExportClientPhoto** element. *Photomap* attributes can be queried using **QueryPhotomap**. *Photomap* can be destroyed using **DestroyPhotomap**.

Error	Cause
Alloc	insufficient resources
IDChoice	invalid *photomap*

A **Photomap** can be populated with a simple **ExecuteImmediate**(**ImportClientPhoto**, **ExportPhotomap**) **Photoflo. PutClientData** is then used to transport the image data. See **ImportClientPhoto** and **ExportPhotomap** and their associated techniques for more information regarding the format of image data.

DestroyPhotomap

XieReqDestroyPhotomap

photomap	XieTypPhotomap

Errors: ***Photomap***
Events: none

DestroyPhotomap disassociates the resource-id from *photomap* and decrements its reference count. If there are no other references, it destroys the **Photomap** identified by *photomap*.

Error	Cause
Photomap	invalid *photomap*

QueryPhotomap

XieReqQueryPhotomap

photomap	XieTypPhotomap
→	
populated	BOOL
data-type	XieTypDataType
data-class	XieTypDataClass
width	XieTypTripletofCARD32
height	XieTypTripletofCARD32
levels	XieTypLevels
decode	XieTypDecodeTechnique

Errors: ***Alloc, Photomap***
Events: none

QueryPhotomap returns the queriable attributes of a **Photomap**.

Photomap is the **Photomap** to be queried.

Populated is a Boolean that indicates that *photomap* has been populated with attributes and data. If *populated* is false, all remaining fields contain zeros. *Data-type* shows whether the number of quantization *levels* is valid (***Constrained***) or unknown (***Unconstrained***). *Data-class* is the class of image data (i.e. ***SingleBand*** or ***TripleBand***). *Width* and *height* are the dimensions of the image data in pixels (per band). *Levels* is the potential dynamic range, or number of quantization levels (per band). *Decode* is the **DecodeTechnique** that will be required to interpret or decompress the data.

If *data-type* is ***Unconstrained***, the returned values for *levels* are zeros. If *data-class* is ***SingleBand***, *width*, *height*, and *levels* for unused bands are returned as zeros.

Error	Cause
FloAlloc	insufficient resources
Photomap	invalid *photomap*

ROI

CreateROI

XieReqCreateROI

roi	XieTypROI

Errors: ***Alloc, IDChoice***
Events: none

CreateROI creates a server **ROI** (*Rectangles-Of-Interest*) resource. A **ROI** resource may be imported into a **Photoflo** and used in conjunction with a **ProcessDomain** specification to restrict processing to a subset of image data. The **ROI**, when populated, will contain a *list-of-rectangles* (of type **Rectangle**).

Roi is the client supplied **ROI** identifier to be assigned to this resource.

The **ROI** is populated (or re-populated) with a *list-of-rectangles* after the successful execution of a **Photoflo** containing an **ExportROI** element which targets *roi*. An **ROI** resource does not have any queriable attributes.

ROI data can be imported into a **Photoflo** using an **ImportROI** element. This data, which is used by several of the **PhotoElements** defined in chapters 7 and 8, can also be exported back to the client with the aid of an **ExportClientROI** element.

Roi can be destroyed using **DestroyROI**.

Error	Cause
Alloc	insufficient resources
IDChoice	invalid *roi*

An **ROI** can be populated with a simple **ExecuteImmediate** (**ImportClientROI**, **ExportROI**) **Photoflo**. **PutClientData** is then used to transport the *list-of-rectangles*.

If the client uses **ExportClientROI** to retrieve a *list-of-rectangles* from the server, the number of rectangles and the content of the list that is returned may differ from original list that was obtained from the client, but its contents will be logically equivalent.

DestroyROI

XieReqDestroyROI

roi	XieTypROI

Errors: ***ROI***
Events: none

DestroyROI disassociates the resource-id from *roi* and decrements its reference count. If there are no other references, it destroys the **ROI** identified by *roi*.

Error	Cause
ROI	invalid *roi*