5

Pipelined Processing

What is a Photoflo?

A **Photoflo** is a directed acyclic graph. Each node has zero or more input connections, and zero or one output connection. A node specifies the source for an input by identifying another upstream node. A node's output connection can be used as a source for any number of downstream input connections. A **Photoflo** is permitted to have taps and multiple final destinations.

The protocol representation of a **Photoflo** is a list of elements. Since the entire list is sent to the server as a single protocol request, the total length of the list, including its protocol header, must fit within a maximum size protocol message (see, *maximum-request-length,* established by X11 connection setup).

Each element of the **Photoflo** is identified by its position in the list. This position, or index, is called a **Phototag**. The first element in the list is index one (1). The order of elements in the list does not have to match the **Photoflo** topology because there is no implicit coupling of output N to input N+1. The source for each element's input connections are explicitly specified using the **Phototag**s of upstream elements.

There are three types of elements. *Import* elements bring data into the **Photoflo** from external resources or the client, have one output connection, and no input connections. *Process* elements perform some operation on the data (e.g. convolution), have one or more input connections, and exactly one output connection. *Export* elements emit data from the **Photoflo** to external resources or to the client, have one input connection, and no output connections. A **Photoflo** should include at least one import element and one export element to be useful.

	Import	**Process**	**Export**
Number of input connections	none	one or more	one
Number of output connections	one	one	none

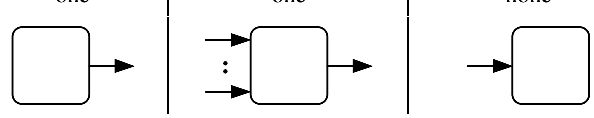

Figure 5-1 Photoflo element input and output connections

All data external to the **Photoflo** (internal server data and client data), are accessed through import and export elements. Therefore, for purposes of **Photoflo** definition and modification, X and XIE resource-ids are considered element parameters rather than element sources or destinations. No element is permitted to reference a **Photoflo** for any reason. It is also an error to specify an export element as an input to any element.

Figure 5-2 illustrates several capabilities of **Photoflo** construction. An `RGB` image is imported from the client. This image is saved in a **Photomap** for later re-use. The blue band is selected and translated (to correct registration). This result is exported back to the client. A **BandCombine** element associates a single band imported from a **Photomap** (the red band), with the green band selected from the initial `RGB` image, and the re-registered blue band. The combined image is then converted to index data and exported to a `PIXMAP` and to a `WINDOW`. The color allocation results are saved in a **ColorList**.

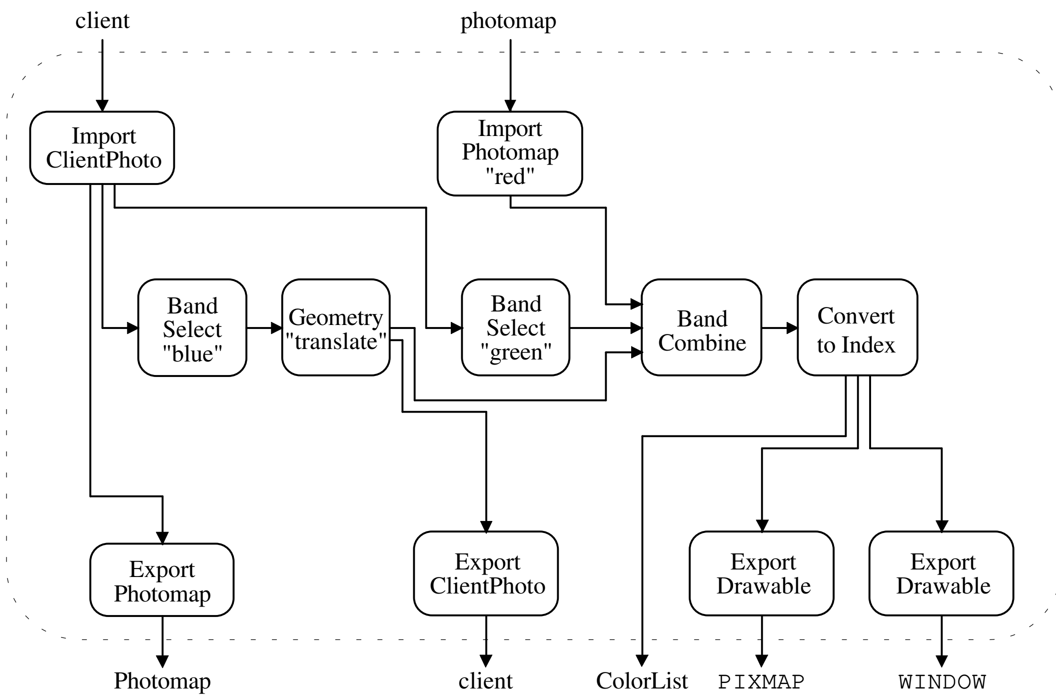

Figure 5-2 Example Photoflo

Two kinds of Photoflos

There are `stored` **Photoflo**s and `immediate` **Photoflo**s. `Stored` **Photoflo**s persist beyond execution and may be modified or totally redefined prior to subsequent executions. `Immediate` **Photoflo**s are ephemeral – a single protocol request defines and begins its execution, then it is automatically destroyed upon completion. Services common to both types of **Photoflo**s are:

- event notification
- error notification
- put data: send data from the client into a Photoflo
- get data: return data exported from a Photoflo back to the client
- query: return information about the Photoflo
- await: block future client requests until the Photoflo completes
- abort: terminate Photoflo execution prematurely

Multi-client Photoflos

A `stored` **Photoflo** can be executed by a client other than the client that created it. It is also possible that the **Photoflo** may reference resources that belong to other clients, and data may be supplied and retrieved by various clients. The following rules apply when multiple clients are involved in the execution of a **Photoflo**:

- errors that stem from executing **Photoflo** elements are sent to the client executing the **Photoflo**,
- errors that stem from executing client data put or get requests go to the client executing the request,
- colors recorded in a **ColorList** resource belong to the client executing the **Photoflo**,
- **PhotofloDone** events are sent to the client executing the **Photoflo**,
- multiple clients can **Await** completion of a given **Photoflo**.

Photoflo States

`Stored` **Photoflo**s enter the ***Inactive*** state upon creation and transition to the ***Active*** state when execution is requested. `Immediate` **Photoflo**s are both created and made ***Active*** by the execute request. A **Photoflo** remains ***Active*** until: all `ImportClient` elements have received a final flag, all export elements have finished their task, and all data exported for the client have been retrieved, or an error prevents further progress, or the client issues an **Abort** request. After execution `stored` **Photoflo**s return to the ***Inactive*** state, whereas `immediate` **Photoflo**s are destroyed (become ***Nonexistent***). The client may destroy a `stored` **Photoflo** at any time.

Figure 5-3 Photoflo states

Flo'ing Data to a Resource

Besides image rendition and display, `immediate` or `stored` **Photoflo**s are also used to populate XIE resources with data and attributes. Some form of processing may be applied to the data, but in the simplest case, a two element import/export **Photoflo** is generally sufficient.

Also all types of ephemeral client data may be imported and used directly by a **Photoflo**, without the necessity of first storing it in an XIE resource. For example, simple transport of an image (compressed or uncompressed) to a `WINDOW` is expected.

Import element	Export element	purpose
ImportClientPhoto	ExportPhotomap	populate a Photomap resource
ImportClientLUT	ExportLUT	populate a LUT resource (lookup table)
ImportClientROI	ExportROI	populate a ROI resource (list-of-rectangles)
ImportClientPhoto	ExportDrawablePlane	display a bitonal image in a WINDOW
ImportClientPhoto	ExportDrawable	display a COLORMAP index image in a WINDOW
ImportPhotomap	ExportDrawablePlane	copy an existing bitonal image to a `PIXMAP`

Table 5-1 Examples of two element Photoflo usage

Name space

For requests that only apply to `stored` **Photoflo**s (Create, Modify, Redefine, Execute, and Destroy), the **Photoflo** is identified solely by its resource-id. For all other requests, events, and errors that reference a **Photoflo**, the **Photoflo** is identified using type **Executable**, which is a tupple identifying the *name-space* and *flo-id* of the **Photoflo**.

Name-space for `stored` **Photoflo**s is always *`ServerIDSpace`* (the value zero), and *flo-id* is the **Photoflo**'s resource-id.

Name-space for `immediate` **Photoflo**s is a **Photospace** resource-id, and *flo-id* is a thirty-two (32) bit value that uniquely identifies the instance of this **Photoflo**.

CreatePhotospace

XieReqCreatePhotospace

name-space	XieTypPhotospace

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

CreatePhotospace defines a *name-space* in which `immediate` **Photoflo**s may be executed. Any client that needs to instantiate `immediate` **Photoflo**s must create at least one **Photospace**.

Name-space is the resource-id for a new **Photospace** that can be used to accommodate `immediate` **Photoflo**s instantiated by this client.

Error	Cause
Alloc	insufficient resources
IDChoice	invalid *name-space*

DestroyPhotospace

XieReqDestroyPhotospace

name-space	XieTypPhotospace

Errors: ***Photospace***
Events: none

DestroyPhotospace will destroy a **Photospace**. Prior to destroying the **Photospace**, all **Photoflo**s that are currently ***Active*** in the **Photospace** will be aborted, exported data pending client retrieval will be freed, and the **Photoflo**s will be destroyed.

Name-space is the **Photospace** to be destroyed.

Error	Cause
Photospace	invalid *name-space*

Immediate Photoflos

ExecuteImmediate

XieReqExecuteImmediate

instance	XieTypExecutable
notify	`BOOL`
element-list	`LISTof`XieTypPhotoElement

Errors: ***FloAlloc, FloID, FloElement, Flo . . .***
Events: **PhotofloDone**

ExecuteImmediate defines and begins execution of an `immediate` **Photoflo**. Execution is asynchronous. The **Photoflo** is destroyed after execution completes and all data exported for the client have been retrieved. It is legal to have multiple unique instances of `immediate` **Photoflo**s (and `stored` **Photoflo**s) ***Active*** concurrently.

Instance specifies the **Photospace/flo-id** tupple by which this **Photoflo** will be identified in other requests, events, or errors. *Notify* specifies whether a **PhotofloDone** event should be sent upon completion. *Element-list* defines the import, process, and export elements to be executed.

If any clients have blocked themselves during the execution of the **Photoflo** (see **Await**), they will become unblocked when *photoflo*'s state changes from ***Active*** to ***Nonexistent***. If *notify* is true, a **PhotofloDone** event is also generated by this transition. Finally, this *instance* is destroyed.

Error	Cause
FloAlloc	insufficient resources
FloID	invalid **Executable** *instance*
FloElement	invalid element-type(s) in *element-list*
Flo . . .	see element descriptions for errors detected by elements in *element-list*

See **ExecutePhotoflo** for a general outline of the execution phases of a **Photoflo**.

Stored Photoflos

The following requests are used to: create, modify, redefine, execute, and destroy `stored` **Photoflo**s. In these requests the **Photoflo** instance is identified only by its resource-id. However, errors and events generated due to these requests identify the instance using type **Executable** (the *name-space* is *`ServerIDSpace`* and the *flo-id* is the **Photoflo**'s resource-id).

CreatePhotoflo

XieReqCreatePhotoflo

photoflo	XieTypPhotoflo
element-list	`LISTof`XieTypPhotoElement

Errors: ***Alloc, IDChoice, FloAlloc, FloElement, Flo . . .***
Events: none

CreatePhotoflo creates a `stored` **Photoflo** resource, defines its complete contents, and sets it in the ***Inactive*** state.

Photoflo is a new resource-id by which this **Photoflo** will be identified in other requests, events, or errors. *Element-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. **CreatePhotoflo** will only store the **Photoflo**'s definition and parameter validation is delayed until an execute request is received.

Error	Cause
Alloc	insufficient resources for *photoflo*
IDChoice	invalid *photoflo*
FloAlloc	insufficient resources for *element-list*
FloElement	invalid element-type(s) in *element-list*
Flo . . .	see element descriptions for errors detected by elements in *element-list*

DestroyPhotoflo

XieReqDestroyPhotoflo

photoflo	XieTypPhotoflo

Errors: ***Photoflo***
Events: none

DestroyPhotoflo will destroy a `stored` **Photoflo**. If *photoflo* was ***Active***, it is aborted and all exported data that are pending client retrieval are freed prior to destroying *photoflo*.

Photoflo is the **Photoflo** to be destroyed.

Error	Cause
Photoflo	invalid *photoflo*

See also **Abort**, which terminates a **Photoflo**'s execution without destroying it.

ExecutePhotoflo

XieReqExecutePhotoflo

photoflo	XieTypPhotoflo
notify	`BOOL`

Errors: ***Photoflo, FloAccess, FloAlloc***
Events: **PhotofloDone**

ExecutePhotoflo 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` **Photoflo**s (and `immediate` **Photoflo**s) ***Active*** concurrently.

Photoflo is the **Photoflo** to be executed. *Notify* specifies whether a **PhotofloDone** event should be sent upon completion.

Conceptually, **Photoflo** execution is broken into at least three phases[1]:

1. Initialization:
   a. the **Photoflo** state is set ***Active***,
   b. bind external inputs (XIE reference counts are incremented)
   c. imported attributes are propagated *downstream*
   d. attributes and element parameter values are validated
   e. lookup external destinations[2]
   f. create receptors for data to be exported to XIE resources[2]
2. Execution:
   a. data from server resources, if any, is pulled into the **Photoflo**
   b. data from the client, if any, is pushed into the **Photoflo** (**PutClientData**)
   c. processed data is exported to server resources as required[2]
   d. processed data is made available for client retrieval (**GetClientData)**[3]
3. Completion:
   a. unbind external inputs
   b. join exported data with associated XIE resources[2] and unbind remaining resources
   c. report any error that occurred
   d. if *notify* is true, send a **PhotofloDone** event
   e. if **Await** has been requested, un-block client execution
   f. the **Photoflo** state is set ***Inactive***

If an error occurs during any phase, the client is notified and the **Photoflo** is terminated. Other events can be sent by individual elements, as specified in their descriptions.

Error	Cause
Photoflo	invalid *photoflo*
FloAccess	attempt to execute *photoflo* when it is already ***Active***
FloAlloc	insufficient resources
Flo . . .	see element descriptions for errors detected by photoflo elements

[1] Optimization, an additional phase, would be roughly associated with the initialization phase.
[2] For **LUT**, **Photomap**, and **ROI** resources refer to page 4-1 "**Binding Resources to Photoflos**".
[3] Completion is contingent upon all such data being retrieved by the client.

ModifyPhotoflo

XieReqModifyPhotoflo

photoflo	XieTypPhotoflo
start	XieTypPhototag
element-list	`LISTof`XieTypPhotoElement

Errors: ***Photoflo, FloAlloc, FloElement, FloSource, Flo . . .***
Events: none

ModifyPhotoflo allows element parameters of a `stored` **Photoflo** to be modified.

Photoflo is the **Photoflo** to be modified. *Start* specifies the **Phototag** where element replacement is to begin. *Element-list* is a sequential list of elements which will replace existing elements.

ModifyPhotoflo allows *parameter* modification only. No topological changes are allowed. Elements cannot be deleted, inserted, or appended.

Error	Cause
Photoflo	invalid *photoflo*
FloAccess	attempt to change *photoflo* while it is ***Active***
FloAlloc	insufficient resources
FloElement	attempt to change element-type(s) in *element-list*
	attempt to append additional element(s) to *photoflo*
FloSource	invalid *start*
	attempt to change **Phototag** input connections in *element-list*
Flo . . .	see element descriptions for errors detected by elements in *element-list*

Multiple **ModifyPhotoflo** requests can be sent in order to edit individual elements, but, for greater efficiency and particularly when repetitive modify/execute requests are expected, elements can be grouped such that a single **ModifyPhotoflo** can perform multiple element modifications.

RedefinePhotoflo

XieReqRedefinePhotoflo

photoflo	XieTypPhotoflo
element-list	`LISTof`XieTypPhotoElement

Errors: ***FloAccess, Photoflo. FloAlloc, FloElement, Flo . . .***
Events: none

RedefinePhotoflo allows all elements of a `stored` **Photoflo** to be removed and replaced with a new list. There are no restrictions on changing element types, **Phototag** sources, or the list's size.

Photoflo is the **Photoflo** to be redefined. *Element-list* is a sequential list of elements which will replace all existing elements.

Error	Cause
Photoflo	invalid *photoflo*
FloAccess	attempt to change *photoflo* while it is ***Active***
FloAlloc	insufficient resources
FloElement	invalid element-type(s) in *element-list*
Flo . . .	see element descriptions for errors detected by elements in *element-list*

RedefinePhotoflo may be considered a hint that the new list of elements is in some way similar to those being replaced. **RedefinePhotoflo** can also be used as a means to conserve resource-ids.

Sending Data to the Server

PutClientData

XieReqPutClientData

instance	XieTypExecutable
element	XieTypPhototag
final	`BOOL`
band-number	`CARD8`
data	XieTypDataStream

Errors: ***FloAlloc, FloAccess, FloID, FloElement, FloValue***
Events: none

PutClientData 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.

Instance and *element* identify the **Photoflo** and specific `ImportClient` element to receive the data. *Final* specifies that this is the last (or only) segment of data to be sent.

If *element* is a band oriented element, *band-number* specifies which client band of data is being sent (*interleave* and *band-order* specified for the `ImportClient` element or technique determine how client bands are mapped to server bands).

Data is a counted list of bytes that comprises the data stream. The organization and contents of the 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*.

Error	Cause
FloAlloc	insufficient resources
FloAccess	**Executable** *instance* not ***Active***
FloID	invalid **Executable** *instance*
FloElement	invalid **Phototag** or element-type specified by *element*
FloValue	invalid *band-number*
	for non-image data, *data* contains a partial aggregate

All types of client data (*list-of-rectangles*, *lookup tables*, and *images*) are transported to the server using **PutClientData**. Tiled images can be transported using multiple **ImportClientPhoto** elements, which in turn feed a **PasteUp** element.

Retrieving Data from the Server

GetClientData

XieReqGetClientData

instance	XieTypExecutable
element	XieTypPhototag
max-bytes	`CARD32`
terminate	`BOOL`
band-number	`CARD8`
→	
new-state	XieTypExportState
data	XieTypDataStream

Errors: ***FloAlloc, FloAccess, FloID, FloElement, FloValue***
Events: none

GetClientData retrieves data from an `ExportClient` element within an ***Active*** **Photoflo**. Data are returned 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 available. The format of the data depends on the parameters given to the `ExportClient` element from which the data are requested.

Instance and *element* identify the **Photoflo** and specific `ExportClient` element from which to retrieve data. *Max-bytes* specifies the maximum number of data bytes that can be sent to the client. *Terminate* is a Boolean that can be used to indicate that no more data are wanted after this request has been satisfied. *Band-number* specifies which client band is to be retrieved (*interleave* and *band-order* parameters specified for the `ExportClient` element technique determine how server bands are mapped to client bands).

New-state indicates the data availability status of the `ExportClient` element after satisfying the current request. *Data* is the counted list of bytes that is returned.

If *terminate* is true, *new-state* will be ***ExportDone*** (the `ExportClient` element will discard any remaining data and stop producing additional data).

If *new-state* is ***ExportEmpty*** and *notify* (for the `ExportClient` element) was specified as ***NewData***, another **ExportAvailable** event will be sent when additional data become availabile.

If the request is sent to an `ExportClient` element that either: does not have any data, was *terminate*d by a previous **GetClientData** request, or has already returned all its data (***ExportDone*** sent), the request will return a zero length *data* 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 (i.e. *max-bytes* is effectively rounded down by the server to the match the nearest aggregate size).

Error	Cause
FloAlloc	insufficient resources
FloAccess	**Executable** *instance* not ***Active***
FloID	invalid **Executable** *instance*
FloElement	invalid **Phototag** or element-type specified by *element*
FloValue	invalid *band-number*

Servers are required to buffer a non-zero amount of data per `ExportClient` element. Beyond that point execution may be suspended until the client retrieves sufficient data.

Terminate may be used to prematurely terminate output from an `ExportClient` element. If *terminate* is not used, all data produced by the `ExportClient` element must be retrieved before the **Photoflo** can leave the ***Active*** state.

Status

QueryPhotoflo

XieReqQueryPhotoflo

instance	XieTypExecutable
→	
state	XieTypPhotofloState
data-expected	`LISTof`XieTypPhototag
data-available	`LISTof`XieTypPhototag

Errors: ***FloAlloc***
Events: none

QueryPhotoflo will return the current status of a **Photoflo**.

Instance identifies the **Photoflo** that is being queried.

State indicates the state of the **Photoflo**. *Data-expected* is a list of `ImportClient` elements that are expecting data from the protocol stream. *Data-available* is a list of `ExportClient` elements from which data for the protocol stream are available. Either or both of these lists may be of length zero.

Specifying an unknown or invalid *instance* will result in a reply *state* of ***nonexistent*** and zero length *data-expected* and *data-available* lists.

Error	Cause
FloAlloc	insufficient resources

Synchronization

Await

XieReqAwait

instance	XieTypExecutable

Errors: ***FloAlloc***
Events: none

Await blocks all further requests for this client connection from being honored by the server while the **Photoflo** is ***Active***. When the **Photoflo** transitions from the ***Active*** state, blocked requests are allowed to be processed in the order received.

Instance identifies the ***Active*** **Photoflo** which is to block requests from the client issuing the **Await**.

If *instance* is invalid or the **Photoflo** is not ***Active*** no action is taken; it is not an error, and the client is not blocked.

Error	Cause
FloAlloc	insufficient resources

If a **Photoflo** has no `ExportClient` elements, the client can call **Await**. If a **Photoflo** has exactly one `ExportClient` element, the client can just read bytes or be event-driven. If a **Photoflo** has multiple `ExportClient` elements, the client should be event-driven.

Warning: calling **Await** before sending all import data (including a *final* flag) 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.

Termination

Abort

XieReqAbort

instance	XieTypExecutable

Errors: none
Events: none

Abort will prematurely terminate execution of a **Photoflo**.

Instance identifies the **Photoflo** which is to be aborted.

Any output from the **Photoflo** that is pending client retrieval is freed. If *instance* is a `stored` **Photoflo** it will return to the ***Inactive*** state. `Immediate` **Photoflo**s are destroyed.

If *instance* is invalid or the **Photoflo** is not ***Active*** no action is taken; it is not an error, and nothing is destroyed.