pixmap-p *object* Function

Returns:
pixmap — Type **boolean**.

Returns true if the argument is a **pixmap** object and **nil** otherwise.

pixmap-plist *pixmap* Function

Returns:
plist — A property list.

Returns and (with **setf**) sets the property list for the specified *pixmap*. This function provides a hook where extensions can add data.

pixmap — A **pixmap**.

Pixmaps

4.8 A *pixmap* is a three-dimensional array of bits. A pixmap is normally thought of as a two-dimensional array of pixels, where each pixel can be a value from 0 to $2^n - 1$, where *n* is the depth of the pixmap. A pixmap can also be thought of as a stack of *n* bitmaps. A *bitmap* is a single bit pixmap of depth 1. CLX provides functions to:

- Create or free a pixmap
- Test if an object is a pixmap
- Test if two pixmap objects are equal
- Return the pixmap resource ID from a **pixmap** object

Note that pixmaps can only be used on the screen where they were created. Pixmaps are off-screen server resources that are used for a number of operations. These include defining patterns for cursors or as the source for certain raster operations.

create-pixmap &key **:width :height :depth :drawable** Function

Returns:
pixmap — Type **pixmap**.

Creates a pixmap of the specified **:width**, **:height**, and **:depth**. It is valid to pass a window whose class is **:input-only** as the **:drawable** argument. The **:width** and **:height** arguments must be nonzero. The **:depth** must be supported by the screen of the specified **:drawable**.

:width, **:height** — The nonzero width and height (type **card16**).

:depth — The depth (type **card8**) of the pixmap.

:drawable — A **drawable** which determines the screen where the pixmap will be used.

free-pixmap *pixmap* Function

Allows the X server to free the pixmap storage when no other server resources reference it. The pixmap should never be referenced again.

pixmap — A **pixmap**.

pixmap-display *pixmap* Function

Returns:
display — Type **display**.

Returns the **display** object associated with the specified *pixmap*.

pixmap — A **pixmap**.

pixmap-equal *pixmap-1 pixmap-2* Function

Returns true if the two arguments refer to the same server resource, and **nil** if they do not.

pixmap-1, *pixmap-2* — A three-dimensional array of bits to be tested.

pixmap-id *pixmap* Function

Returns:
id — Type **resource-id**.

Returns the unique resource ID that has been assigned to the specified *pixmap*.

pixmap — A **pixmap**.

map-subwindows *window* Function

Maps all child windows for a specified *window* in top-to-bottom stacking order. The X server generates an **:exposure** event on each newly visible window. This function is much more efficient than mapping each child individually.

window — A **window**.

unmap-window *window* Function

Unmaps the specified *window* and causes the X server to generate an **:unmap-notify** event. If the specified *window* is already unmapped, **unmap-window** has no effect. Normal exposure processing on formerly obscured windows is performed. Any child window is no longer viewable. Unmapping the *window* generates **:exposure** events on windows that were formerly obscured by *window* and its children.

window — A **window**.

unmap-subwindows *window* Function

Unmaps all child windows for the specified *window* in bottom to top stacking order. The X server generates an **:unmap-notify** event on each child and **:exposure** events on formerly obscured windows. Using this function is much more efficient than unmapping child windows individually.

window — A **window**.

Destroying Windows

4.7 CLX provides functions to destroy a window or destroy all children of a window. Note that by default, windows are destroyed when a connection is closed. For further information, see paragraph 2.4, Closing the Display, and paragraph 12.4, Client Termination.

destroy-window *window* Function

Destroys the specified *window* as well as all of its inferiors. The windows should never again be referenced. If the specified *window* is mapped, it is automatically unmapped. The window and all of its inferiors are then destroyed, and a **:destroy-notify** event is generated for each window. The ordering of the **:destroy-notify** events is such that for any given window being destroyed, **:destroy-notify** is generated on the window's inferiors before being generated on the window. The ordering among siblings and across sub-hierarchies is not otherwise constrained. If the *window* is a root window, no windows are destroyed. Destroying a mapped window generates **:exposure** events on other windows that the mapped window obscured.

window — A **window**.

destroy-subwindows *window* Function

Destroys all inferiors of the specified *window*, in bottom to top stacking order. The X server generates a **:destroy-notify** event for each window. This is much more efficient than deleting many windows individually. The inferiors should never be referenced again.

window — A **window**.

A subwindow will appear on the screen as long as all of its ancestors are mapped and not obscured by a sibling or clipped by an ancestor. Mapping a window that has an unmapped ancestor does not display the window, but marks it as eligible for display when the ancestor becomes mapped. Such a window is called unviewable. When all its ancestors are mapped, the window becomes viewable and remains visible on the screen if not obscured by any sibling or ancestor.

Any output to a window not visible on the screen is discarded. **:exposure** events are generated for the window when part or all of it becomes visible on the screen. A client only receives the **:exposure** events if it has selected them. Mapping or unmapping a window does not change its stacking order priority.

map-window *window* Function

Maps the *window*. This function has no effect when the *window* is already mapped.

If the override-redirect attribute of the *window* is **:off** and another client has selected **:substructure-redirect** on the parent window, the X server generates a **:map-request** event and the **map-window** function does not map the *window*. Otherwise, the *window* is mapped, and the X server generates a **:map-notify** event.

If the *window* becomes visible and no earlier contents for it are remembered, **map-window** tiles the window with its background. If no background was defined for the window, the existing screen contents are not altered, and the X server generates one or more **:exposure** events. If a backing-store was maintained while the window was unmapped, no **:exposure** events are generated. If a backing-store will now be maintained, a full window exposure is always generated. Otherwise, only visible regions may be reported. Similar tiling and exposure take place for any newly viewable inferiors.

map-window generates **:exposure** events on each **:input-output** window that it causes to become visible.

window — A **window**.

After reparenting the specified *window,* a **:reparent-notify** event is generated. The override-redirect attribute of the *window* is passed on in this event. Window manager clients normally should ignore this event if this attribute is **:on**. See Section 12, Events and Input, for more information on **:reparent-notify** event processing. Finally, if the specified *window* was originally mapped, a **map-window** operation is automatically performed on it.

The X server performs normal exposure processing on formerly obscured windows. It might not generate **:exposure** events for regions from the initial **unmap-window** operation if they are immediately obscured by the final **map-window** operation.

It is an error if any of the following are true:

- The new *parent* window is not on the same screen as the old parent window.
- The new *parent* window is the specified *window* or an inferior of the specified *window*.
- The specified *window* has a **:parent-relative** background attribute and the new *parent* window is not the same depth as the specified *window*.

window — A **window**.

parent — The new parent **window**.

x, *y* — The position (type **int16**) of the *window* in its new *parent*. These coordinates are relative to the *parent*'s origin, and specify the new position of the upper, left, outer corner of the *window*.

translate-coordinates *source source-x source-y destination* Function

Returns:
destination-x — Type **int16** or **null**.
destination-y — Type **int16** or **null**.
destination-child — Type **window** or **null**.

Returns the position defined by *source-x* and *source-y* (relative to the origin of the *source* window), expressed as coordinates relative to the origin of the *destination* window.

source — A **window** defining the source coordinate system.

source-x, *source-y* — Coordinates (**int16**) relative to the origin of the *source* **window**.

destination — A **window** defining the destination coordinate system.

Mapping Windows

4.6 A window is considered mapped if a **map-window** call has been made on it. When windows are first created, they are not mapped because an application may wish to create a window long before it is mapped to the screen. A mapped window may not be visible on the screen for one of the following reasons:

- It is obscured by another opaque sibling window.
- One of its ancestors is not mapped.
- It is entirely clipped by an ancestor.

If some other client has selected **:substructure-redirect** on the *window*, a **:circulate-request** event is generated, and no further processing is performed. Otherwise, the child window is lowered and a **:circulate-notify** event is generated if the *window* is actually restacked.

window — A **window**.

circulate-window-up *window* Function

Raises the lowest mapped child of the specified *window* that is partially or completely occluded by another child to the top of the stack. Completely unobscured children are unaffected. Exposure processing is performed on formerly obscured windows.

If another client has selected **:substructure-redirect** on the *window*, a **:circulate-request** event is generated, and no further processing is performed. Otherwise, the child window is raised and a **:circulate-notify** event is generated if the *window* is actually restacked.

window — A **window**.

Window Hierarchy

4.5 All the windows in X are arranged in a strict hierarchy. At the top of the hierarchy are the root windows, which cover the display screens. Each root window is partially or completely covered by its child windows. All windows, except for root windows, have parents. Child windows can have their own children. In this way, a tree of arbitrary depth on each screen can be created. CLX provides several functions for examining and modifying the window hierarchy.

drawable-root *drawable* Function

Returns:
root-window — Type **window**.

Returns the root window of the specified *drawable*.

drawable — A **drawable**.

query-tree *window* &key (**:result-type 'list**) Function

Returns:
children — Type **sequence** of **window**.
parent — Type **window** or **null**.
root — Type **window**.

Returns the *children* windows, the *parent* window, and the *root* window for the specified *window*. The children are returned as a sequence of windows in current stacking order, from bottom-most (first) to top-most (last). The **:result-type** specifies the type of children sequence returned.

window — A **window**.

:result-type — A valid type specifier for a sub-type of **sequence**. The default is a **list**.

reparent-window *window parent x y* Function

Changes a *window*'s *parent* within a single screen. There is no way to move a window between screens.

The specified *window* is reparented by inserting it as a child of the specified *parent*. If the *window* is mapped, an **unmap-window** operation is automatically performed on the specified *window*. The *window* is then removed from its current position in the hierarchy and inserted as the child of the specified *parent*. The *window* is placed on top in the stacking order with respect to sibling windows.

Table 4-1 **Groups of Reader and Setf Functions**

Group	Reader Functions	Setf Functions
Window Attributes	**window-all-event-masks** **window-backing-pixel** **window-backing-planes** **window-backing-store** **window-bit-gravity** **window-class** **window-colormap** **window-colormap-installed-p** **window-do-not-propagate-mask** **window-event-mask** **window-gravity** **window-map-state** **window-override-redirect** **window-save-under** **window-visual**	**window-background** **window-backing-pixel** **window-backing-planes** **window-backing-store** **window-bit-gravity** **window-border** **window-colormap** **window-cursor** **window-do-not-propagate-mask** **window-event-mask** **window-gravity** **window-override-redirect** **window-save-under**
Drawable Geometry	**drawable-border-width** **drawable-depth** **drawable-height** **drawable-root** **drawable-width** **drawable-x** **drawable-y**	**drawable-border-width** **drawable-height** **drawable-width** **drawable-x** **drawable-y** **window-priority**

The results from a sequence of calls to **setf** functions in a given group are cached and sent in a single server request, either upon exit from the *body* or when a reader function from the corresponding group is called.

with-state sends a single request to update all its cached values for the *drawable* before the first call to a reader function within the *body* and also before the first call to a reader function following a sequence of calls to **setf** functions from the corresponding group.

drawable — A **display**.

body — The forms in which attributes accesses are batched.

Stacking Order

4.4 Sibling windows can *stack* on top of each other. Windows above can *obscure* or *occlude* lower windows. This relationship between sibling windows is known as the stacking order. The **window-priority** function can be used to change the stacking order of a single window. CLX also provides functions to raise or lower children of a window. Raising a mapped window can generate **:exposure** events for the window and any mapped subwindows that were formerly obscured. Lowering a mapped window can generate **:exposure** events on any windows it formerly obscured.

circulate-window-down *window* Function

Lowers the highest mapped child of the specified *window* that partially or completely occludes another child to the bottom of the stack. Completely unobscured children are unaffected. Exposure processing is performed on formerly obscured windows.

All window attributes can be returned or changed in a single request. Similarly, all drawable geometry values can be returned or changed in a single request. **with-state** combines accesses to these values into the minimum number of server requests necessary to guarantee that each read access returns the current server state of the *drawable*. The number of server requests sent depends on the sequence of calls to reader and **setf** functions within the dynamic extent of the *body*. There are two groups of reader and **setf** functions—the Window Attributes group and the Drawable Geometry group—as shown in Table 4-1.

window-p *object* Function

Returns:
window-p — Type **boolean**.

Returns non-**nil** if the *object* argument is a window; otherwise, it returns **nil**.

window-plist *window* Function

Returns:
plist — A property list.

Returns and (with **setf**) sets the property list for the specified *window*. This function provides a hook where extensions can hang data.

window — A **window**.

setf (**window-priority** *window*) (&optional *sibling*) *mode* Function

Returns:
mode — One of **:above**, **:below**, **:bottom-if**, **:opposite**, or **:top-if**.

Changes the stacking priority element of the *window* to the specified value. It is an error if the *sibling* argument is specified and is not actually a sibling of the window. Note that the priority of an existing window cannot be returned from the X server.

When changing the priority of a window, if the override-redirect attribute of the window is **:off** and some other client has selected :substructure-redirect on the parent, a :configure-request event is generated, and no further processing is performed. Otherwise, the priority is changed.

window — A **window**.

sibling — An optional argument specifying that *window* is to be restacked relative to this sibling **window**.

mode — One of **:above**, **:below**, **:bottom-if**, **:opposite**, or **:top-if**.

window-save-under *window* Function

Returns:
save-under — Either **:on** or **:off**.

Returns and (with **setf**) changes the value of the save-under attribute for the specified *window*. Changing the save-under attribute of a mapped window may have no immediate effect.

window — A **window**.

window-visual *window* Function

Returns:
visual-type — Type **card29**.

Returns the *visual-type* associated with the specified *window*.

window — A **window**.

with-state *drawable* &body *body* Macro

Batches successive read and write accesses to window attributes and drawable geometry, in order to minimize the number of requests sent to the server. Batching occurs automatically within the dynamic extent of the *body*. The *body* is not executed within a **with-display** form.

For example, suppose the size of the window's parent is changed by *W* pixels in width and *H* pixels in height. The following table shows, for each possible gravity value, the resulting change in the window's position relative to its parent's origin. When the window is moved, two events are generated—a **:configure-notify** event followed by a **:gravity-notify** event.

Gravity	X Change	Y Change
:center	*W*/2	*H*/2
:east	*W*	*H*/2
:north	*W*/2	0
:north-east	*W*	0
:north-west	0	0
:south	*W*/2	*H*
:south-east	W	H
:south-west	0	H
:west	0	H/2

A **:static** gravity indicates that the position of the window should not move relative to the origin of the root window.

An **:unmap** gravity is like **:north-west**, except the window is also unmapped and an **:unmap-notify** event is generated. This **:unmap-notify** event is generated after the **:configure-notify** event is generated for the parent.

window — A **window**.

window-id *window* Function

Returns:
The **resource-id** of the window.

Returns the unique ID assigned to *window*.

window — A **window**.

window-map-state *window* Function

Returns:
map-state — One of **:unmapped**, **:unviewable**, or **:viewable**.

Returns the map state of *window*. A window is **:unviewable** if it is mapped but some ancestor is unmapped.

window — A **window**.

window-override-redirect *window* Function

Returns:
override-redirect — Either **:on** or **:off**.

Returns and (with **setf**) changes the value of the override-redirect attribute for *window*. The default override-redirect of a new window is **:off**.

The override-redirect attribute determines whether or not attempts to change window geometry or parent hierarchy can be *redirected* by a window manager or some other client. The functions that might be affected by the override-redirect attribute are **circulate-window-down**, **circulate-window-up**, **drawable-border-width**, **drawable-height**, **drawable-width**, **drawable-x**, **drawable-y**, **map-window**, and **window-priority**.

window — A **window**.

setf (**window-cursor**) *window cursor* Function

Returns:
cursor — Type **cursor** or **:none**.

Changes the *cursor* attribute of the *window* to the specified value. Changing the cursor of a root window to **:none** restores the default cursor. Note that the cursor of window cannot be returned from the X server. The default cursor of a new window is **:none**.

window — A **window**.

cursor — Either **cursor** or **:none**.

window-display *window* Function

Returns:
display — Type **display**.

Returns the **display** object associated with the specified *window*.

window — A **window**.

window-do-not-propagate-mask *window* Function

Returns:
do-not-propagate-mask — Type **mask32**.

Returns and (with **setf**) changes the do-not-propagate-mask attribute for the window. The default do-not-propagate-mask of a new window is zero.

If a window receives an event from one of the user input devices, and if no client has selected to receive the event, the event can instead be propagated up the window hierarchy to the first ancestor for which some client has selected it. However, any event type selected by the do-not-propagate-mask is not be propagated. The types of events that can be selected by the do-not-propagate-mask are those of type **device-event-mask-class**. See paragraph 12.2, Selecting Events.

window — A **window**.

window-equal *window-1 window-2* Function

Returns:
boolean.

Returns non-**nil** if the two arguments are the same window, and **nil** if they are not.

window-1, *window-2* — The windows to compare for equality.

window-event-mask *window* Function

Returns:
event-mask — Type **mask32**.

Returns and (with **setf**) changes the value of the event-mask attribute for the *window*. The default event-mask of a new window is zero.

window — A **window**.

window-gravity *window* Function

Returns:
gravity — Type **win-gravity**.

Returns and (with **setf**) changes the gravity attribute of the *window*. If a parent window is reconfigured without changing its inside width or height, then all child windows move with the parent and are not changed. Otherwise, each child of the resized parent is moved, depending on the child's gravity attribute. The default gravity of a new window is **:north-west**.

A server can choose to ignore the specified bit-gravity attribute and use **:forget** instead. A **:forget** bit-gravity attribute indicates that the window contents are always discarded after a size change, even if backing-store or save-under attributes are **:on**. The window's background is displayed (unless it is **:none**), and zero or more **:exposure** events are generated.

window — A **window**.

setf (**window-border**) *window border* Function

Returns:
border — Either a **pixel**, a **pixmap**, or **:copy**.

Changes the *border* attribute of the *window* to the specified value. This operation is not allowed on an **:input-only** window. Changing the border attribute also causes the window border to be repainted. Note that the border of a window cannot be returned from the X server. The default border of a new window is **:copy**.

A border pixmap is tiled to fill the border. The border pixmap tile origin is the same as the background tile origin. A border pixmap and the window must have the same root and depth. If the border is **:copy**, the parent's border is copied and used; subsequent changes to the parent's border do not affect the window border.

window — A **window**.

border — Either a **pixel**, a **pixmap**, or **:copy**.

window-class *window* Function

Returns:
class — Either **:input-output** or **:input-only**.

Returns the *class* of the specified *window*.

window — A **window**.

window-colormap *window* Function

Returns:
colormap — Type **colormap** or **null**.

Returns and (with **setf**) changes the value of the colormap attribute for the specified *window*. A value of **:copy** is never returned, since the parent's colormap attribute is actually copied, but the attribute can be set to **:copy** in a **setf** form. Changing the colormap of a window (defining a new map, not changing the contents of the existing map) generates a **:colormap-notify** event. Changing the colormap of a visible window may have no immediate effect on the screen (see **install-colormap**). The default colormap of a new window is **:copy**.

window — A **window**.

window-colormap-installed-p *window* Function

Returns:
colormap-installed-p — Type **boolean**.

Returns non-**nil** if the colormap associated with this *window* is installed. Otherwise, this function returns **nil**.

window — A **window**.

window-backing-pixel *window* Function

Returns:
backing-pixel — Type **pixel**.

Returns and (with **setf**) changes the value of the backing-pixel attribute for the specified *window*. Changing the backing-pixel attribute of a mapped window may have no immediate effect. The default backing-pixel of a new window is zero.

window — A **window**.

window-backing-planes *window* Function

Returns:
backing-planes — Type **pixel**.

Returns and (with **setf**) changes the value of the backing-planes attribute for the specified *window*. Changing the backing-planes attribute of a mapped window may have no immediate effect. The default backing-planes of a new window is all one's.

window — A **window**.

window-backing-store *window* Function

Returns:
backing-store-type — One of **:always**, **:not-useful**, or **:when-mapped**.

Returns and (with **setf**) changes the value of the backing-store attribute for the specified *window*. Changing the backing-store attribute of an obscured window to **:when-mapped** or **:always** may have no immediate effect. The default backing-store of a new window is **:not-useful**.

window — A **window**.

window-bit-gravity *window* Function

Returns:
bit-gravity — Type **bit-gravity**.

Returns and (with **setf**) changes the bit-gravity attribute of the *window*. If a window is reconfigured without changing its inside width or height, the contents of the window move with the window and are not lost. Otherwise, the contents of the resized window are either moved or lost, depending on its bit-gravity attribute. The default bit-gravity of a new window is **:forget**.

For example, suppose a window's size is changed by *W* pixels in width and *H* pixels in height. The following table shows, for each bit-gravity value, the change in position (relative to the window origin) that results for each pixel of the window contents.

Bit-Gravity	X Change	Y Change
:center	*W*/2	*H*/2
:east	*W*	*H*/2
:north	*W*/2	0
:north-east	*W*	0
:north-west	0	0
:south	*W*/2	*H*
:south-east	W	H
:south-west	0	H
:west	0	H/2

A **:static** bit-gravity indicates the contents or window should not move relative to the origin of the root window.

When the inside size of the window is changed, the children of the window can move according to their window gravity. Depending on the window's bit gravity, the contents of the window can also be moved.

drawable — A **drawable** object.

drawable-x *drawable* Function

Returns:
outside-left — Type **int16**.

drawable-y *drawable* Function

Returns:
outside-top — Type **int16**.

These functions return the x or y coordinate of the specified *drawable*. They always return zero if the *drawable* is a pixmap. These coordinates define the location of the top left pixel of the window's border or of the window, if it has no border. Used with **setf**, these functions also change the x or y coordinate of a window. However, the x or y coordinate of a pixmap cannot be changed.

Changing the x and y coordinates moves a window without changing its size or stacking priority. Moving a mapped window generates **:exposure** events on any formerly obscured windows.

When changing the position of a window, if the override-redirect attribute of the window is **:off** and some other client has selected **:substructure-redirect** on the parent, a **:configure-request** event is generated, and no further processing is performed. Otherwise, the window is moved.

drawable — A **drawable** object.

window-all-event-masks *window* Function

Returns:
all-event-masks — Type **mask32**.

Returns the inclusive-or of the event masks selected on the specified *window* by all clients.

window — A **window**.

setf (**window-background**) *window background* Function

Returns:
background — Either a **pixel**, a **pixmap**, **:none**, or **:parent-relative**.

Changes the *background* attribute of the *window* to the specified value. This operation is not allowed on an **:input-only** window. Changing the background does not cause the window contents to be changed. Note that the background of a window cannot be returned from the X server. The default background of a new window is **:none**.

In general, the server automatically fills in exposed areas of the window when they are first made visible. A background pixmap is tiled to fill each area. However, if the background is **:none**, the server will not modify exposed areas. If the background is **:parent-relative**, the window and its parent must have the same depth. In this case, the window shares the same background as its parent. The parent's background is not copied and is reexamined whenever the window's background is required. If the background is **:parent-relative**, the background pixmap tile origin is the same as the parent's; otherwise, the tile origin is the window origin.

window — A **window**.

background — Either a **pixel**, a **pixmap**, **:none**, or **:parent-relative**.

Window Attributes

4.3 The following paragraphs describe the CLX functions used to return or change window attributes. Using the **with-state** macro improves the performance of attribute access by batching related accesses in the minimum number of server requests.

drawable-border-width *drawable* Function

Returns:
border-width — Type **card16**.

Returns the *border-width* of the *drawable* in pixels. It always returns zero if the *drawable* is a pixmap or an **:input-only** window. Used with **setf**, this function also changes the border width of the **:input-only** window. The default border width of a new window is zero.

Changing just the border width leaves the outer left corner of a window in a fixed position but moves the absolute position of the window's origin. It is an error to make the border width of an **:input-only** window nonzero.

When changing the border-width of a window, if the override-redirect attribute of the window is **:off** and some other client has selected **:substructure-redirect** on the parent, a **:configure-request** event is generated, and no further processing is performed. Otherwise, the border-width is changed.

drawable — A **drawable** object.

drawable-depth *drawable* Function

Returns:
depth — Type **card8**.

Returns the depth of the specified *drawable* (bits per pixel).

drawable — A **drawable** object.

drawable-height *drawable* Function

Returns:
inside-height — Type **card16**.

drawable-width *drawable* Function

Returns:
inside-width — Type **card16**.

These functions return the height or width of the *drawable*. These coordinates define the inside size of the *drawable*, in pixels. Used with **setf**, these functions also change the inside height or width of a window. However, the height or width of a pixmap cannot be changed.

Changing the width and height resizes a window without changing its position or stacking priority.

Changing the size of a mapped window may cause the window to lose its contents and generate an **:exposure** event. If a mapped window is made smaller, **:exposure** events are generated on windows that it formerly obscured.

When changing the size of a window, if the override-redirect attribute of the window is **:off** and some other client has selected **:substructure-redirect** on the parent, a **:configure-request** event is generated, and no further processing is performed. Otherwise, if another client has selected **:resize-redirect** on the window, a **:resize-request** event is generated, and the current inside width and height are maintained. Note that the override-redirect attribute of the window has no effect on **:resize-redirect** and that **:substructure-redirect** on the parent has precedence over **:resize-redirect** on the window.

The **:class** of a window can be **:input-output** or **:input-only**. Windows of class **:input-only** cannot be used as the destination drawable for graphics output and can never receive **:exposure** events, but otherwise operate the same as **:input-output** windows. The **:class** can also be **:copy**, in which case the new window has the same class as its **:parent**.

For an **:input-output** window, the **:visual** and **:depth** must be a combination supported by the **:parent**'s screen, but the **:depth** need not be the same as the **:parent**'s. The **:parent** of an **:input-output** window must also be **:input-output**. A **:depth** of 0 means that the depth of the **:parent** is used.

For an **:input-only** window, the **:depth** must be zero, and the **:visual** must be supported by the **:parent**'s screen. The **:parent** of an **:input-only** window can be of any class. The only attributes that can be given for an **:input-only** window are **:cursor**, **:do-not-propagate-mask**, **:event-mask**, **:gravity**, and **:override-redirect**.

:parent — The parent window. This argument is required.

:x, **:y** — **int16** coordinates for the outside upper-left corner of the new window with respect to the origin (inside upper-left corner) of the **:parent**. These arguments are required.

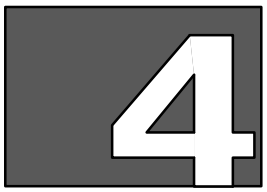
:width, **:height** — **card16** values for the size of the new window. These arguments are required.

:depth — A **card16** specifying the depth of the new window.

:class — One of **:input-output**, **:input-only**, or **:copy**.

:visual — A **card29** ID specifying the visual type of the new window.

:background, **:backing-pixel**, **:backing-planes**, **:backing-store**, **:bit-gravity**, **:border**, **:border-width**, **:colormap**, **:cursor**, **:do-not-propagate-mask**, **:event-mask**, **:gravity**, **:override-redirect**, **:save-under** — Initial attribute values for the new window. If **nil**, the default value is defined by the X protocol. See paragraph 4.3, Window Attributes.



WINDOWS AND PIXMAPS

Drawables

4.1 Both windows and pixmaps can be used as sources and destinations in graphics operations. These are collectively known as *drawables*. The following functions apply to both windows and pixmaps.

drawable-display *drawable* Function

Returns the display for the specified *drawable*.

drawable — A **drawable** object.

drawable-equal *drawable-1 drawable-2* Function

Returns true if the two arguments refer to the same server resource, and **nil** if they do not.

drawable-1, *drawable-2* — **drawable** objects.

drawable-id *drawable* Function

Returns:
id — Type **resource-id**.

Returns the unique resource ID assigned to the specified *drawable*.

drawable — A **drawable** object.

drawable-p *drawable* Function

Returns:
boole — Type **boolean**.

Returns true if the argument is a **drawable** and **nil** otherwise.

drawable-plist *drawable* Function

Returns:
plist — A property list.

Returns and (with **setf**) sets the property list for the specified *drawable*. This function provides a hook where extensions can add data.

Creating Windows

4.2 A window is a **drawable** that can also receive input events. CLX represents a window with a **window** object. The **create-window** function creates a new **window** object.

create-window &key **:parent :x :y :width :height** (**:depth** 0) Function
(**:border-width** 0) (**:class :copy**) (**:visual :copy**) **:background**
:border :gravity :bit-gravity :backing-store :backing-planes :backing-pixel
:save-under :event-mask :do-not-propagate-mask :override-redirect :colormap
:cursor
Returns:
window — Type **window**.

Creates and returns a window. A **:parent** window must be specified; the first window created by a client will have a root window as its **:parent**. The new window is initially unmapped and is placed on top of its siblings in the stacking order. A **:create-notify** event is generated by the server.