- **:sync-keyboard** — If the keyboard is frozen and actively grabbed by the client, keyboard event processing continues normally until the next **:key-press** or **:key-release** event is reported to the client, at which time the keyboard again appears to freeze. However if the reported event causes the keyboard grab to be released, the keyboard does not freeze. **:sync-keyboard** has no effect if the keyboard is not frozen by the client, or if the keyboard is not grabbed by the client.

- **:replay-keyboard** — If the keyboard is actively grabbed by the client and is frozen as the result of an event having been sent to the client (either from the activation of a grab-key, or from a previous **allow-events** with mode **:sync-keyboard**, but not from a **grab-keyboard**), the keyboard grab is released and that event is completely reprocessed, but this time ignoring any passive grabs at or above (towards the root) the **grab-window** of the grab just released. The request has no effect if the keyboard is not grabbed by the client, or if the keyboard is not frozen as the result of an event.

- **:sync-both** — If both pointer and keyboard are frozen by the client, event processing (for both devices) continues normally until the next **:button-press**, **:button-release**, **:key-press**, or **:key-release** event is reported to the client for a grabbed device (button event for the pointer, key event for the keyboard). At this time, the devices again appear to freeze. If the reported event causes the grab to be released, the devices do not freeze. However, if the other device is still grabbed, then a subsequent event for it will still cause both devices to freeze. **:sync-both** has no effect unless both pointer and keyboard are frozen by the client. If the pointer of keyboard is frozen twice by the client on behalf of two separate grabs, **:sync-both** *thaws* for both, but a subsequent freeze for **:sync-both** will only freeze each device once.

- **:async-both** — If the pointer and the keyboard are frozen by the client, event processing for both devices continues normally. If a device is frozen twice by the client on behalf of two separate grabs, **:async-both** *thaws* for both. **:async-both** has no effect unless both pointer and keyboard are frozen by the client.

:async-pointer, **:sync-pointer**, and **:replay-pointer** have no effect on processing of keyboard events. **:async-keyboard**, **:sync-keyboard**, and **:replay-keyboard** have no effect on processing of pointer events.

It is possible for both a pointer grab and a keyboard grab to be active simultaneously by the same or different clients. When a device is frozen on behalf of either grab, no event processing is performed for the device. It is possible for a single device to be frozen due to both grabs. In this case, the freeze must be released on behalf of both grabs before events can again be processed.

display — A **display**.

mode — One of: **:async-pointer**, **:sync-pointer**, **:reply-pointer**, **:async-keyboard**, **:sync-keyboard**, **:replay-keyboard**, **:async-both**, **:sync-both**.

time — A **timestamp**.

The *event-codes* argument gives the event type keyword for the event described. If several event types share the same slots, then *event-codes* can be a list of event type keywords. *slot-declarations* is a list containing an element for each event data item. The order of *slot-declarations* corresponds to the order of event data items defined by the X protocol.

Each element of *slot-declarations* is a list of the form (*type slot-name**), where *type* is a Common Lisp type specifier and *slot-name* is a slot name symbol. The effect of such a list is to declare that the next data items in the event have the given data *type* and are associated with the given *slot-name* symbols. *slot-name* can also be a list of slot name symbols; in this case, each symbol in the list is an alias that refers to the same event data item.

event-codes — An event type keyword or a list of event type keywords.

slot-declarations — A list of clauses defining event slot symbols.

Releasing Queued Events

12.13 A client grabbing the keyboard or pointer can freeze the reporting of events on that device. When an input device is thus frozen, the server queues events until explicitly requested to release them by the grabbing client. CLX programs can use the **allow-events** function to release queued events from a frozen input device.

allow-events *display mode* &optional *time* Function

Releases some queued events if the client has caused a device to freeze. The request has no effect if the *time* is earlier than the last-grab time of the most recent active grab for the client, or if the *time* is later than the current server time. If *time* is **nil**, the current server time is used. The effect of this function depends on the specified *mode*.

- **:async-pointer** — If the pointer is frozen by the client, pointer event processing continues normally. If the pointer is frozen twice by the client on behalf of two separate grabs, **:async-pointer** releases events for both grab*s*. **:async-pointer** has no effect if the pointer is not frozen by the client, but the pointer need not be grabbed by the client.

- **:sync-pointer** — If the pointer is frozen and actively grabbed by the client, pointer event processing continues normally until the next **:button-press** or **:button-release** event is reported to the client, at which time the pointer again appears to freeze. However, if the reported event causes the pointer grab to be released, the pointer does not freeze. **:sync-pointer** has no effect if the pointer is not frozen by the client, or if the pointer is not grabbed by the client.

- **:replay-pointer** — If the pointer is actively grabbed by the client and is frozen as the result of an event having been sent to the client (either from the activation of a **grab-button**, or from a previous **allow-events** with mode **:sync-pointer**, but not from a **grab-pointer**), the pointer grab is released and that event is completely reprocessed, but this time ignoring any passive grabs at or above (towards the root) the **grab-window** of the grab just released. The request has no effect if the pointer is not grabbed by the client, or if the pointer is not frozen as the result of an event.

- **:async-keyboard** — If the keyboard is frozen by the client, keyboard event processing continues normally. If the keyboard is frozen twice by the client on behalf of two separate grabs, **:async-keyboard** releases events for both grabs. **:async-keyboard** has no effect if the keyboard is not frozen by the client, but the keyboard need not be grabbed by the client.

1. Convert the current *selection* value to the *target* type.
2. Store the converted selection value in the *property*. If *property* is **nil**, then the owner should choose the *property*.
3. Call **send-event** to send a **:selection-notify** event to the *requestor* containing the *property* with the converted value. If the *selection* could not be converted to the *target* type, then a **nil** *property* should be sent. The **:selection**, **:target**, and **:time** arguments to **send-event** should be the same as those received in the **:selection-request** event. The event-mask argument to **send-event** should be **nil**; that is, the **:selection-notify** event should be sent to client that created the *requestor*.

NOTE: Standard conventions for inter-client communication require the following additional steps in processing a **:selection-request** event:

1. The property used to store the selection value must belong to the requestor.
2. If the property is **nil**, the target type **atom** should be used as the property name.
3. If the window did not actually own the selection at the given time, the request should be refused, just as if it could not be converted to the target type.

---

window, event-window — Type **window**.

The selection owner receiving the event.

requestor — Type **window**.

The window requesting the converted *selection*.

selection — Type **keyword**.

The selection to be converted.

target — Type **keyword**.

An **xatom** specifying the type of the converted *selection* value.

property — Type **(or null keyword)**.

A requestor window property.

time — Type **timestamp**.

The timestamp sent in the client **convert-selection** request.

Declaring Event Types

12.12.8 CLX uses the **declare-event** macro to define the event slot symbols that access the contents of X events. Most client applications do not need to use **declare-event** because the declarations for all core X events are already defined by CLX. Programmers using extensions to the X protocol can use **declare-event** to allow CLX to handle new event types returned by an extended X server.

declare-event *event-codes* &rest *slot-declarations* Macro

Defines a mapping between event slot symbols and the data items in event messages received from an X server.

window, event-window — Type **window**.

The window losing ownership of the *selection*.

selection — Type **keyword**.

The name of the selection.

time — Type **timestamp**.

The last-change time recorded for the *selection*.

:selection-notify Event Type

The **:selection-notify** event is sent to a client calling **convert-selection**. **:selection-notify** reports the result of the client request to return the current value of a *selection* into a particular form. **:selection-notify** is sent using **send-event** by the owner of the selection or (if no owner exists) by the X server. A client can neither select **:selection-notify** events nor avoid receiving them.

NOTE: Standard conventions for inter-client communication require the following additional steps in processing a **:selection-notify** event:

1. The client receiving this event should call **get-property** to return the converted selection value.
2. After receiving the selection value, the property should then be deleted (either by using the **:delete-p** argument to **get-property** or by calling **delete-property**).

---

window, event-window — Type **window**.

The requestor window given in the call to **convert-selection**.

selection — Type **keyword**.

The selection to be converted.

target — Type **keyword**.

An **xatom** specifying the type of the converted selection value. This is the same target type given in the call to **convert-selection**.

property — Type **(or null keyword)**.

The window property containing the converted selection. If the property is **nil**, then either the *selection* has no owner or the owner could not perform the conversion to the *target* type.

time — Type **timestamp**.

The timestamp from the client call to **convert-selection**.

:selection-request Event Type

The **:selection-request** event is reported to the owner of a selection when a client calls **convert-selection**. This event requests the selection owner to convert the current value of a *selection* into a specified form and to return it to the requestor. A client can neither select **:selection-request** events nor avoid receiving them.

The selection owner should respond to a **:selection-request** event by performing the following steps:

Client Communications Events

12.12.7 The client communications events discussed in the following paragraphs are: **:client-message**, **:property-notify**, **:selection-clear**, **:selection-request**, and **:selection-notify**.

:client-message Event Type

The **:client-message** event is generated exclusively by client calls to **send-event**. The X server places no interpretation on the *type* or content of *data* sent in a **:client-message**. A client can neither select **:client-message** events nor avoid receiving them.

window, event-window — Type **window**.

The window receiving the event.

type — Type **keyword**.

An xatom keyword that specifies the type of client message. Interpretation of the type is determined solely by agreement between the sending and receiving clients.

format — Type (**member 8 16 32**).

An integer that specifies whether *data* should be viewed as a sequence of 8-bit, 16-bit, or 32-bit quantities.

data — Type **(sequence integer)**.

The data content of the client message. *data* always consists of 160 bytes — depending on format, either 20 8-bit values, 10 16-bit values or 5 32-bit values. The amount of this data actually used by a particular client message depends on the type.

:property-notify Event Type

Selected by: — **:property-change**.

The **:property-notify** event is generated when a window property is changed or deleted.

window, event-window — Type **window**.

The window receiving the event.

atom — Type **keyword**.

The property that was changed or deleted.

state — Type (**member :new-value :deleted**).

state is **:new-value** if the property was changed using **change-property** or **rotate-properties**, even if zero-length data was added or if all or part of the property was replaced with identical data. *state* is **:deleted** if the property was deleted using **delete-property** or **get-property**.

time — Type **timestamp**.

The server time when the property was changed or deleted.

:selection-clear Event Type

The **:selection-clear** event is reported to the previous owner of a *selection* when the owner of the *selection* is changed. The selection owner is changed by a client using **setf**. A client can neither select **:selection-clear** events nor avoid receiving them.

width, height — Type **card16**.

width and *height* specify the requested size of the *window* interior. If either *width* or *height* is not specified in the *value-mask*, then it is set to the current window size.

border-width — Type **card16**

The requested *window* border width. If *border-width* is not specified in the *value-mask*, then it is set to the current window *border-width*.

stack-mode — Type **(member :above :below :top-if :bottom-if :opposite)**.

stack-mode and *above-sibling* specify the requested stacking priority of the *window*. If *stack-mode* is not specified in the *value-mask*, then it is set to **:above**.

above-sibling — Type (**or null window**).

stack-mode and *above-sibling* specify the requested stacking priority of the *window*. If *above-sibling* is not specified in the *value-mask*, then it is set to **nil**.

value-mask — Type **mask16**.

Specifies the changed *window* attributes contained in the redirected client request. Each 1 bit specifies that the corresponding attribute was changed.

:map-request Event Type

Selected by: — **:substructure-redirect** on parent.

The **:map-request** event is generated when a client application maps a *window* that has the override-redirect attribute **:off**.

parent, event-window — Type **window**.

The window receiving the event. The receiving client must have selected **:substructure-redirect** on this window.

window — Type **window**.

The window to be mapped.

:resize-request Event Type

Selected by: — **:resize-redirect**.

The **:resize-request** event is generated when a client program sets the *width* or *height* attributes of a *window* that has the override-redirect attribute **:off**.

w*indow, event-window* — Type **window**.

The window to be resized.

width, height — Type **card16**.

width and *height* specify the requested size of the w*indow* interior. If either *width* or *height* was unchanged in the client request, then it is set to the current window size.

place — Type **(member :top :bottom)**.

The new stacking priority requested for *window*.

:colormap-notify Event Type

Selected by: — **:colormap-change**.

The **:colormap-notify** event is generated with *new-p* **t** when the *colormap* associated with a *window* is changed, installed, or uninstalled.

window, event-window — Type **window**.

The window receiving the event.

colormap — Type **(or null colormap)**.

The colormap attribute of the window.

new-p — Type **boolean**.

If *new-p* is true, then the *window*'s colormap attribute has changed to the given *colormap*. Otherwise, the *window*'s colormap attribute has not, but the *colormap* has been installed or uninstalled.

installed-p — Type **boolean**.

If *installed-p* is true, then the *colormap* is currently installed.

:configure-request Event Type

Selected by: — **:substructure-redirect** on parent.

The **:configure-request** event is generated when a client program sets the *x*, *y*, *width*, *height*, *border-width* or stacking priority attributes of a window that has the override-redirect attribute **:off**.

parent, event-window — Type **window**.

The window receiving the event. The receiving client must have selected **:substructure-redirect** on this window.

window — Type **window**.

The window to be reconfigured.

x — Type **int16**.

x and *y* specify the requested upper-left corner position of the *window* relative to the parent. If either *x* or *y* is not specified in the value-mask, then it is set to the current window position.

y — Type **int16**.

x and *y* specify the requested upper-left corner position of the *window* relative to the *parent*. If either *x* or *y* is not specified in the *value-mask*, then it is set to the current window position.

:visibility-notify Event Type

Selected by: — **:visibility-change**.

The **:visibility-notify** event is sent when the visibility of a *window* changes. **:visibility-notify** events are never generated on **:input-only** windows. For the purposes of this event, the visibility of the *window* is not affected by its subwindows.

All **:visibility-notify** events caused by a hierarchy change are generated after any hierarchy event caused by that change (for example, **:unmap-notify**, **:map-notify**, **:configure-notify**, **:gravity-notify**, or **:circulate-notify**). Any **:visibility-notify** event on a given window is generated before any **:exposure** events on that window, but it is not required that all **:visibility-notify** events on all windows be generated before all **:exposure** events on all windows. The ordering of **:visibility-notify** events with respect to **:focus-out**, **:enter-notify**, and **:leave-notify** events is not constrained.

window, event-window — Type **window**.

The window that changed in visibility.

state — Type (**member :unobscured :partially-obscured :fully-obscured**).

When the *window* was either unviewable or it was viewable and at least partially obscured, and the *window* changed to viewable and completely unobscured, then *state* is **:unobscured**.

When the *window* was either unviewable or it was viewable and completely obscured, and the *window* changed to viewable and partially obscured, then *state* is **:partially-obscured**.

When the *window* was either unviewable or it was at least partially visible, and the *window* changed to viewable and completely obscured, then *state* is **:fully-obscured**.

Structure Control Events

12.12.6 The following paragraphs describe events used to *redirect* client requests that reconfigure, restack, or map a window. Structure control events are typically used only by window managers and not by ordinary client applications. Structure control events report redirected requests, allowing a window manager to modify the requests before they are actually performed. However, if the override-redirect attribute of a window is **:on**, then no requests are redirected and no structure control events are generated.

:circulate-request Event Type

The **:circulate-request** event is generated when a client application calls **circulate-window-up** or **circulate-window-down** with a window that has the override-redirect attribute **:off**. The *window* argument specifies the window to be restacked, and *place* specifies what the new position in the stacking order should be (either **:top** or **:bottom**).

Selected by: — **:substructure-redirect** on *parent*.

parent, event-window — Type **window**.

The window receiving the event. The receiving client must have selected **:substructure-redirect** on this window.

window — Type **window**.

The window to be restacked.

:reparent-notify Event Type

Selected by: — **:structure-notify** on a window or **:substructure-notify** on its old or new parent.

The **:reparent-notify** event is generated when a *window* is reparented.

event-window — Type **window**.

The window receiving the event.

window — Type **window**.

The window that was reparented.

parent — Type **window**.

The new parent of the *window*.

x — Type **int16**.

x and y specify the upper-left corner position of the *window* relative to its new *parent*.

y — Type **int16**.

x and y specify the upper-left corner position of the *window* relative to its new *parent*.

override-redirect-p — Type **boolean**.

override-redirect-p is true if the override-redirect attribute of the *window* is **:on**; otherwise, it is **nil**. See **window-override-redirect** in paragraph 4.3, Window Attributes.

:unmap-notify Event Type

Selected by: — **:structure-notify** on a window or **:substructure-notify** on its parent.

The **:unmap-notify** event is generated when a mapped *window* is unmapped.

event-window — Type **window**.

The window receiving the event.

window — Type **window**.

The window that was unmapped.

co*nfigure-p* — Type **boolean**.

configure-p is true if the *window* has a win-gravity attribute of **:unmap**, and the event was generated because *window*'s parent was resized.

:gravity-notify Event Type

Selected by: — **:structure-notify** on a window or **:substructure-notify** on its parent.

The X server can report **:gravity-notify** events to clients wanting information about when a *window* is moved because of a change in the size of its parent. The X server generates this event whenever a client application actually moves a child window as a result of resizing its parent by calling **with-state** with the appropriate arguments set.

event-window — Type **window**.

The window receiving the event.

window — Type **window**.

The window that was moved.

x — Type **int16**.

x and y specify the new upper-left corner position of the *window* relative to its parent.

y — Type **int16**.

x and y specify the new upper-left corner position of the *window* relative to its parent.

:map-notify Event Type

Selected by: — **:structure-notify** on a window or **:substructure-notify** on its parent.

The X server can report **:map-notify** events to clients wanting information about which windows are mapped. The X server generates this event type whenever a client application changes the *window*'s state from unmapped to mapped by calling **map-window** or **map-subwindow**.

To receive this event type, you **setf :structure-notify** as the event-mask on the *window*'s **event-mask** slot. You can also receive this event type by **setf**ing the **:substructure-notify** event-mask on the parent window.

event-window — Type **window**.

The window receiving the event.

window — Type **window**.

The window that was mapped.

override-redirect-p — Type **boolean**.

override-redirect-p is true if the override-redirect attribute of the *window* is **:on**; otherwise, it is **nil**. See **window-override-redirect** in paragraph 4.3, Window Attributes.

:create-notify Event Type

Selected by: **:substructure-notify**.

The **:create-notify** event is generated when a *window* is created and is sent to the *parent* window.

parent, event-window — Type **window**.

The parent window receiving the event.

window — Type **window**.

The new window created.

x — Type **int16**.

x and *y* specify the initial upper-left corner position of the *window* relative to the parent.

y — Type **int16**.

x and *y* specify the initial upper-left corner position of the *window* relative to the parent.

width — Type **card16**.

width and *height* specify the initial size of the *window* interior.

height — Type **card16**.

width and *height* specify the initial size of the *window* interior.

border-width — Type **card16**.

The initial *window* border width.

override-redirect-p — Type **boolean**.

override-redirect-p is true if the override-redirect attribute of the *window* is **:on**; otherwise, it is **nil**. See **window-override-redirect** in paragraph 4.3, Window Attributes.

:destroy-notify Event Type

Selected by — **:structure-notify** on a window or **:substructure-notify** on its parent.

The **:destroy-notify** event is generated when a *window* is destroyed. The ordering of the **:destroy-notify** events is such that for any given window, **:destroy-notify** is generated on all inferiors of a window before **:destroy-notify** is generated on the *window*. The ordering among siblings and across subhierarchies is not otherwise constrained.

event-window — Type **window**.

The window receiving the event.

window — Type **window**.

The window that was destroyed.

:configure-notify Event Type

Selected by: — **:structure-notify** on a window or **:substructure-notify** on its parent.

The **:configure-notify** event is generated when the position or size of a window actually changes as a result of a client application setting its *x*, *y*, *width*, *height*, or *border-width* attributes.

event-window — Type **window**.

The window receiving the event.

window — Type **window**.

The window that was moved or resized.

x — Type **int16**.

x and *y* specify the new upper-left corner position of the *window* relative to its parent.

y — Type **int16**.

x and *y* specify the new upper-left corner position of the *window* relative to its parent.

width — Type **card16**.

width and *height* specify the new size of the *window* interior.

height — Type **card16**.

width and *height* specify the new size of the *window* interior.

border-width — Type **card16**.

The new *window* border width.

above-sibling — Type (**or null window**).

The sibling immediately below the *window*. If above-sibling is **nil**, then the *window* is below all of its siblings.

override-redirect-p — Type **boolean**.

override-redirect-p is true if the override-redirect attribute of the *window* is **:on**; otherwise, it is **nil**. See **window-override-redirect** in paragraph 4.3, Window Attributes.

The X server can report **:create-notify** events to clients wanting information about creation of windows. The X server generates this event whenever a client application creates a window by calling **create-window**.

To receive this event type in a client application, you **setf** the **:substructure-notify** as the event-mask in the parent window's event-mask slot.

The destination drawable for the **copy-area** or **copy-plane** function.

major — Type **card8**.

The major opcode for the graphics request generating the event (62 for **copy-area**, 63 for **copy-plane**).

minor — Type **card16**.

The minor opcode for the graphics request generating the event (0 for both **copy-area** and **copy-plane**).

Window State Events

12.12.5 The following paragraphs describe the events that can be received when a window becomes:

- Created
- Destroyed
- Invisible
- Mapped
- Moved
- Reparented
- Resized
- Restacked
- Unmapped
- Visible

:circulate-notify Event Type

Selected by: — **:structure-notify** on a window or **:substructure-notify** on its parent.

A **:circulate-notify** event is generated whenever a window is actually restacked as a result of a client application calling **circulate-window-up** or **circulate-window-down**.

event-window — Type **window**.

The window receiving the event.

window — Type **window**.

The window that was restacked.

place — Type (**member :top :bottom**).

If place is **:top**, the *window* is now on top of all siblings. Otherwise, it is below all siblings.

height — Type **card16**.

The height of the region to redisplay.

count — Type **card16**.

If count is zero, then no more **:exposure** events for this window follow. If count is nonzero, then at least that many more **:exposure** events for this window follow (and possibly more).

:graphics-exposure Event Type

A **:graphics-exposure** event is generated by a call to **copy-area** or **copy-plane** when the exposures attribute of the graphics context is **:on**. A **:graphics-exposure** event reports a destination region whose content cannot be computed because the content of the corresponding source region has been lost. For example, the missing source region may be obscured or may lie outside the current source drawable size. For a given action causing **:graphics-exposure** events, the set of events for a given destination are guaranteed to be reported contiguously.

drawable, event-window — Type **drawable**.

The destination drawable for the **copy-area** or **copy-plane** function.

x — Type **card16**.

The position of the left edge of the destination region, relative to the *drawable*.

y — Type **card16**.

The position of the top edge of the destination region, relative to the *drawable*.

width — Type **card16**.

The width of the destination region.

height — Type **card16**.

The height of the destination region.

count — Type **card16**.

If count is zero then no more **:graphics-exposure** events for the *drawable* follow. If count is nonzero then at least that many more **:graphics-exposure** events for the *drawable* follow (and possibly more).

major — Type **card8**.

The major opcode for the graphics request generating the event (62 for **copy-area**, 63 for **copy-plane**).

minor — Type **card16**.

The minor opcode for the graphics request generating the event (0 for both **copy-area** and **copy-plane**).

:no-exposure Event Type

A **:no-exposure** event is generated by a call to **copy-area** or **copy-plane** when the exposures attribute of the graphics context is **:on**. If no **:graphics-exposure** events are generated, then a single **:no-exposure** event is sent.

drawable, event-window — Type **drawable**.

Exposure Events

12.12.4 The X server cannot guarantee that a window's content is preserved when the window is obscured or reconfigured. X requires client applications to be capable of restoring the contents of a previously-invisible window region whenever it is exposed. Therefore, the X server sends events describing the exposed window and its exposed region. For a simple window, a client can choose to redraw the entire content whenever any region is exposed. For a complex window, a client can redraw only the exposed region.

:exposure Event Type

Selected by: — **:exposure**.

An **:exposure** event is sent when redisplay is needed for a window region whose content has been lost. Redisplay is needed when one of the following occurs:

- A region is exposed for a window and the X server has no backing store for the region
- A region of a viewable window is obscured and the X server begins to honor the window's backing-store attribute of **:always** or **:when-mapped**
- The X server begins to honor an unviewable window's backing-store attribute of **:always** or :**when-mapped**.

The regions needing redisplay are decomposed into an arbitrary set of rectangles, and an **:exposure** event is generated for each rectangle. For a given action causing **:exposure** events, the set of events for a given window are guaranteed to be reported contiguously.

:exposure events are never generated for **:input-only** windows.

All **:exposure** events caused by a hierarchy change are generated after any hierarchy event (**:unmap-notify**, **:map-notify**, **:configure-notify**,**:gravity-notify**, or **:circulate-notify**) caused by that change. All **:exposure** events on a given window are generated after any **:visibility-notify** event on that window, but it is not required that all **:exposure** events on all windows be generated after all visibility events on all windows. The ordering of :**exposure** events with respect to **:focus-out**, **:enter-notify**, and **:leave-notify** events is not constrained.

window, event-window — Type **window**.

The window needing redisplay.

x — Type **card16**.

The position of the left edge of the region to redisplay, relative to the *event-window*.

y — Type **card16**.

The position of the top edge of the region to redisplay, relative to the *event-window*.

width — Type **card16**.

The width of the region to redisplay.

Keyboard and Pointer State Events

12.12.3 The keyboard and pointer state events are **:keymap-notify** and **:mapping-notify.**

:keymap-notify Event Type

Selected by: — :**keymap-state**.

The **:keymap-notify** event returns the current state of the keyboard. **:keymap-notify** is generated immediately after every **:enter-notify** and **:focus-in**.

window, event-window — Type **window**.

The window receiving an **:enter-notify** or **:focus-in** event.

keymap — Type (**bit-vector 256**).

A bit-vector containing the logical state of the keyboard. Each bit set to 1 indicates that the corresponding key is currently pressed. The vector is represented as 32 bytes. For *n* from 0 to 7, byte *n* (from 0) contains the bits for keys 8*n* to 8*n*+7, with the least significant bit in the byte representing key 8*n*.

:mapping-notify Event Type

The X server reports **:mapping-notify** events to all clients. There is no mechanism to express disinterest in this event. The X server generates this event type whenever a client application calls one of the following:

- **set-modifier-mapping** to indicate which keycodes to use as modifiers (the status reply must be **:mapping-success**)
- **change-keyboard-mapping** to change the keyboard mapping
- s**et-pointer-mapping** to set the pointer mapping (the status reply must be **:mapping-success**)

request — Type (**member :modifier :keyboard :pointer**).

Indicates the kind of change that occurred—**:modifier** for a successful **set-modifier-mapping**, **:keyboard** for a successful **change-keyboard-mapping**, and **:pointer** for a successful **set-pointer-mapping**.

start — Type **card8**.

If request is **:keyboard**, then *start* and *count* indicate the range of altered keycodes.

count — Type **card8**.

If request is **:keyboard**, then *start* and *count* indicate the range of altered keycodes.

When the focus moves from **:pointer-root** (or **:none**) to window A, with the pointer in window P:

- If the old focus is **:pointer-root**, **:focus-out** with *kind* **:pointer** is generated on each window from P up to and including P's root (in order)
- **:focus-out** with *kind* **:pointer-root** (or **:none**) is generated on all root windows
- If A is not a root window, **:focus-in** with *kind* **:nonlinear-virtual** is generated on each window from A's root down to but not including A (in order)
- **:focus-in** with *kind* **:nonlinear** is generated on A
- If P is an inferior of A, **:focus-in** with *kind* **:pointer** is generated on each window below A down to and including P (in order)

When the focus moves from **:pointer-root** to **:none** (or vice versa), with the pointer in window P:

- If the old focus is **:pointer-root**, **:focus-out** with *kind* **:pointer** is generated on each window from P up to and including P's root (in order)
- **:focus-out** with *kind* **:pointer-root** (or **:none**) is generated on all root windows
- **:focus-in** with *kind* **:none** (or **:pointer-root**) is generated on all root windows
- If the new focus is **:pointer-root**, **:focus-in** with *kind* **:pointer** is generated on each window from P's root down to and including P (in order)

When a keyboard grab activates (but before generating any actual **:key-press** event that activates the grab), with *G* the **grab-window** for the grab and *F* the current focus, then **:focus-in** and **:focus-out** events with mode **:grab** are generated (as for **:normal** above) as if the focus were to change from *F* to *G*.

When a keyboard grab deactivates (but after generating any actual **:key-release** event that deactivates the grab), with *G* the **grab-window** for the grab and *F* the current focus, then **:focus-in** and **:focus-out** events with mode **:ungrab** are generated (as for **:normal** above) as if the focus were to change from *G* to *F*.

When the focus moves from window A to window B, with window C being their least common ancestor, and with the pointer in window P:

- If P is an inferior of A, **:focus-out** with *kind* **:pointer** is generated on each window from P up to but not including A (in order)
- **:focus-out** with *kind* **:nonlinear** is generated on A
- **:focus-out** with *kind* **:nonlinear-virtual** is generated on each window between A and C exclusive (in that order)
- **:focus-in** with *kind* **:nonlinear-virtual** is generated on each window between C and B exclusive (in that order)
- :focus-in with *kind* **:nonlinear** is generated on B
- If P is an inferior of B, **:focus-in** with *kind* **:pointer** is generated on each window below B down to and including P (in order)

When the focus moves from window A to window B, on different screens, with the pointer in window P:

- If P is an inferior of A, **:focus-out** with *kind* **:pointer** is generated on each window from P up to but not including A (in order)
- **:focus-out** with *kind* **:nonlinear** is generated on A
- If A is not a root window, **:focus-out** with *kind* **:nonlinear-virtual** is generated on each window above A up to and including its root (in order)
- If B is not a root window, **:focus-in** with *kind* **:nonlinear-virtual** is generated on each window from B's root down to but not including B (in order)
- **:focus-in** with *kind* **:nonlinear** is generated on B
- If P is an inferior of B, **:focus-in** with *kind* **:pointer** is generated on each window below B down to and including P (in order)

When the focus moves from window A to **:pointer-root** (or **:none**), with the pointer in window P:

- If P is an inferior of A, **:focus-out** with *kind* **:pointer** is generated on each window from P up to but not including A (in order)
- **:focus-out** with *kind* **:nonlinear** is generated on A
- If A is not a root window, **:focus-out** with *kind* **:nonlinear-virtual** is generated on each window above A up to and including its root (in order)
- **:focus-in** with *kind* **:pointer-root** (or **:none**) is generated on all root windows
- If the new focus is **:pointer-root**, **:focus-in** with *kind* **:pointer** is generated on each window from P's root down to and including P (in order)

mode — Type **(member :normal :while-grabbed :grab :ungrab)**.

Events generated by **set-input-focus** when the keyboard is not grabbed have mode **:normal**. Events generated by **set-input-focus** when the keyboard is grabbed have mode **:while-grabbed**. Events generated when a keyboard grab activates have mode **:grab**, and events generated when a keyboard grab deactivates have mode **:ungrab**.

kind — Type (**member :ancestor :virtual :inferior :nonlinear :nonlinear-virtual :pointer :pointer-root :none**).

When the focus moves from window A to window B, and A is an inferior of B, with the pointer in window P:

- **:focus-out** with *kind* **:ancestor** is generated on A
- **:focus-out** with *kind* **:virtual** is generated on each window between A and B exclusive (in that order)
- **:focus-in** with *kind* **:inferior** is generated on B
- If P is an inferior of B, but P is not A or an inferior of A or an ancestor of A, **:focus-in** with *kind* **:pointer** is generated on each window below B down to and including P (in order)

When the focus moves from window A to window B, and B is an inferior of A, with the pointer in window P:

- If P is an inferior of A, but P is not A or an inferior of B or an ancestor of B, **:focus-out** with *kind* **:pointer** is generated on each window from P up to but not including A (in order)
- **:focus-out** with *kind* **:inferior** is generated on A
- **:focus-in** with *kind* **:virtual** is generated on each window between A and B exclusive (in that order)
- **:focus-in** with *kind* **:ancestor** is generated on B

When a pointer grab deactivates (but after generating any actual **:button-release** event that deactivates the grab), with *G* the **grab-window** for the grab and *P* the window the pointer is in, then **:enter-notify** and **:leave-notify** events with mode **:ungrab** are generated (as for **:normal** above) as if the pointer were to suddenly warp from from some position in *G* to its current position in *P*. However, the pointer does not warp, and the current pointer position is used as both the *initial* and *final* positions for the events.

focus-p — Type **boolean**.

If *event-window* is the focus window or an inferior of the focus window, then *focus-p* is **t**; otherwise, *focus-p* is **nil**.

state — Type **card16**.

A mask that gives the state of the buttons and modifier keys just before the event.

time — Type **card32**.

A timestamp for the moment when the event occurred.

root — Type **window**.

The root window containing the final pointer position.

root-x — Type **int16**.

The x coordinate of the pointer position relative to root at the time of the event.

root-y — Type **int16**.

The y coordinate of the pointer position relative to root at the time of the event.

child — Type (**or null window**).

In a **:leave-notify** event, if a child of the *event-window* contains the initial position of the pointer, the *child* slot is set to that child; otherwise, the *child* slot is **nil**. For an **:enter-notify** event, if a child of the *event-window* contains the final pointer position, the *child* slot is set to that child; otherwise, the *child* slot is **nil**.

same-screen-p — Type **boolean**.

True if *event-window* and root are on the same screen.

Input Focus Events

12.12.2 The input focus events are **:focus-in** and **:focus-out**.

:focus-in, :focus-out Event Type

Selected by: — **:focus-change**.

:focus-in and **:focus-out** events are generated when the input focus changes. All **:focus-out** events caused by a window **:unmap** are generated after any **:unmap-notify** event, but the ordering of **:focus-out** with respect to generated **:enter-notify**, **:leave-notify**, **:visibility-notify**, and **:expose** events is not constrained.

window, event-window — Type **window**.

For **:focus-in**, the new input focus window. For **:focus-out,** the previous input focus window.

kind — Type (**member :ancestor :virtual :inferior :nonlinear :nonlinear-virtual**).

When the pointer moves from window A to window B, and A is an inferior of B:

- **:leave-notify** with *kind* **:ancestor** is generated on A
- **:leave-notify** with *kind* **:virtual** is generated on each window between A and B exclusive (in that order)
- **:enter-notify** with *kind* **:inferior** is generated on B

When the pointer moves from window A to window B, and B is an inferior of A:

- **:leave-notify** with *kind* **:inferior** is generated on A
- **:enter-notify** with *kind* **:virtual** is generated on each window between A and B exclusive (in that order)
- **:enter-notify** with *kind* **:ancestor** is generated on B

When the pointer moves from window A to window B, with window C being their least common ancestor:

- **:leave-notify** with *kind* **:nonlinear** is generated on A
- **:leave-notify** with *kind* **:nonlinear-virtual** is generated on each window between A and C exclusive (in that order)
- **:enter-notify** with *kind* **:nonlinear-virtual** is generated on each window between C and B exclusive (in that order)
- **:enter-notify** with *kind* **:nonlinear** is generated on B

When the pointer moves from window A to window B, on different screens:

- **:leave-notify** with *kind* **:nonlinear** is generated on A
- If A is not a root window, **:leave-notify** with *kind* **:nonlinear-virtual** is generated on each window above A up to and including its root (in order)
- If B is not a root window, **:enter-notify** with *kind* **:nonlinear-virtual** is generated on each window from B's root down to but not including B (in order)
- **:enter-notify** with *kind* **:nonlinear** is generated on B

When a pointer grab activates (but after any initial warp into a confine-to window, and before generating any actual **:button-press** event that activates the grab), with *G* the **grab-window** for the grab and *P* the window the pointer is in, then **:enter-notify** and **:leave-notify** events with mode **:grab** are generated (as for **:normal** above) as if the pointer were to suddenly warp from its current position in *P* to some position in *G*. However, the pointer does not warp, and the pointer position is used as both the *initial* and *final* positions for the events.

:motion-notify events are generated only when the motion begins and ends in the window. The granularity of motion events is not guaranteed, but a client selecting for motion events is guaranteed to get at least one event when the pointer moves and comes to rest. Selecting **:pointer-motion** generates **:motion-notify** events regardless of the state of the pointer buttons. By selecting some subset of **:button[1-5]-motion** instead, **:motion-notify** events are only received when one or more of the specified buttons are pressed. By selecting **:button-motion**, **:motion-notify** events are only received when at least one button is pressed. If **:pointer-motion-hint** is also selected, the server is free to send only one **:motion-notify**, until either the key or button state changes, the pointer leaves the event window, or the client calls **query-pointer** or **motion-events**.

hint-p — Type **boolean**.

True if the event is a hint generated by selecting **:pointer-motion-hint**.

:enter-notify, :leave-notify Event Type

Selected by: — **:enter-window** or **:leave-window**.

If pointer motion or a window hierarchy change causes the pointer to be in a different window than before, **:enter-notify** and **:leave-notify** events are generated instead of a **:motion-notify** event. All **:enter-notify** and **:leave-notify** events caused by a hierarchy change are generated after any hierarchy event (**:unmap-notify**, **:map-notify**, **:configure-notify**, **:gravity-notify**, or **:circulate-notify**) caused by that change, but the ordering of **:enter-notify** and **:leave-notify** events with respect to **:focus-out**, **:visibility-notify**, and **:exposure** events is not constrained by the X protocol. An **:enter-notify** or **:leave-notify** event can also be generated when a client application calls **change-active-pointer-grab**, **grab-pointer**, or **ungrab-pointer**.

window, event-window — Type **window**.

The window receiving the event.

x — Type **int16**.

The final pointer position. If *event-window* is on the same screen as root, then *x* and *y* are the pointer coordinates relative to the *event-window*; otherwise *x* and *y* are zero.

y — Type **int16**.

The final pointer position. If *event-window* is on the same screen as root, then *x* and *y* are the pointer coordinates relative to the *event-window*; otherwise *x* and *y* are zero.

mode — Type (**member :normal :grab :ungrab**).

Events caused when the pointer is actively grabbed have mode **:grab**. Events caused when an active pointer grab deactivates have mode **:ungrab**. In all other cases, mode is **:normal**.

x — Type **int16**.

If *event-window* is on the same screen as root, then *x* and *y* are the pointer coordinates relative to the *event-window*; otherwise *x* and *y* are zero.

y — Type **int16**.

If *event-window* is on the same screen as root, then *x* and *y* are the pointer coordinates relative to the *event-window*; otherwise *x* and *y* are zero.

state — Type **card16**.

A mask that gives the state of the buttons and modifier keys just before the event.

time — Type **card32**.

A timestamp for the moment when the event occurred.

root — Type **window**.

The root window of the source window.

root-x — Type **int16**.

The x coordinate of the pointer position relative to root at the time of the event.

root-y — Type **int16**.

The y coordinate of the pointer position relative to root at the time of the event.

child — Type (**or null window**).

If the source window is an inferior of the *event-window*, *child* is set to the child of *event-window* that is an ancestor of (or is) the source window; otherwise, it is set to **nil**.

same-screen-p — Type **boolean**.

True if *event-window* and root are on the same screen.

:motion-notify Event Type

Selected by: — **:button-1-motion**, **:button-2-motion**, **:button-3-motion**, **:button-4-motion**, **:button-5-motion**, **:button-motion**, or **:pointer-motion**.

The **:motion-notify** event is generated when the pointer moves. A **:motion-notify** event has the same slots as **:button-press, :button-release**, **:key-press**, and **:key-release** events, with the exception that the *code* slot is replaced by the *hint-p* slot. As with these other events, the event window for **:motion-notify** is found by starting with the source window and looking up the hierarchy for the first window on which any client has selected interest in the event (provided no intervening window prohibits event generation by including **:motion-notify** in its do-not-propagate-mask).The actual window used for reporting can be modified by active grabs.

Keyboard and Pointer Events

12.12.1 The keyboard and pointer events are: **:key-press, :key-release, :button-press, :button-release**, **:motion-notify**, **:enter-notify**, and **:leave-notify.**

:key-press, :key-release, :button-press, :button-release Event Type

Selected by: — **:key-press**, **:key-release**, **:button-press**, or **:button-release**.

:key-press, and **:key-release** events are generated when a key or pointer button changes state. Note that **:key-press** and **:key-release** are generated for all keys, even those mapped to modifiers. All of these event types have the same slots. The window containing the pointer at the time of the event is referred to as the *source* window. The *event window* is the window to which the event is actually reported. The event window is found by starting with the source window and looking up the hierarchy for the first window on which any client has selected interest in the event (provided no intervening window prohibits event generation by including the event type in its do-not-propagate-mask). The actual window used for reporting can be modified by active grabs and, in the case of keyboard events, can be modified by the focus window.

A **:button-press** event has the effect of a temporary **grab-button**. When a pointer button is pressed and no active pointer grab is in progress, the ancestors of the source window are searched from the *root* down, looking for a passive grab to activate. If no matching passive grab on the button exists, then an active grab is started automatically for the client receiving the **:button-press** event, and the last-pointer-grab time is set to the current server time. The effect is essentially equivalent to calling **grab-button** with the following arguments:

Argument	Description
window	The event window.
button	The button that was pressed.
event-mask	The client's selected pointer events on the event window.
:modifiers	0
:owner-p	**t** if the client has **:owner-grab-button** selected on the event window; otherwise **nil**.
:sync-pointer-p	**nil**
:sync-keyboard-p	**nil**
:confine-to	**nil**
:cursor	**nil**

The **:button-press** grab is terminated automatically when all buttons are released. The functions **ungrab-pointer** and **change-active-pointer-grab** can both be used to modify the **:button-press** grab.

window, event-window — Type **window**.

The window receiving the event.

code — Type **card8**.

The *code* argument varies with the event type. For **:key-press** and **:key-release**, *code* is the keycode (see paragraph 14.4, Keyboard Encodings). For **:button-press** and **:button-release**, *code* is the pointer button number.

A zero modifier mask is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). It is not required that all specified modifiers have currently assigned keycodes. A *key* of **:any** is equivalent to issuing the request for all possible keycodes. Otherwise, the *key* must be in the range specified by **display-min-keycode** and **display-max-keycode** in the connection setup.

window — A **window**.

key — The key (type **card8**) to be grabbed or **:any**.

:modifiers — A **modifier-mask**.

:owner-p — If true, all client windows receive keyboard input normally.

:sync-pointer-p — Indicates whether the pointer is in synchronous or asynchronous mode.

:sync-keyboard-p — Indicates whether the keyboard is in synchronous or asynchronous mode.

:time — A **timestamp**.

ungrab-key *window key* &key (**:modifiers** 0) Function

Releases the *key* combination on the specified *window* if it was grabbed by this client. A zero modifier mask of **:any** is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). A *key* of **:any** is equivalent to issuing the request for all possible keycodes. **ungrab-key** has no effect on an active grab.

window — A **window**.

key — The key (type **card8**) to be released or **:any**.

:modifiers — A **modifier-mask**.

Event Types

12.12 The following paragraphs contain detailed descriptions of the contents of each event type. In CLX, events are not actually represented by structures, but rather by lists of keyword values passed to handler functions or by values bound to symbols within the clauses of **event-case** and **event-cond** forms. Nevertheless, it is convenient to describe event contents in terms of slots and to identify the components of events with slot name symbols. In fact, CLX uses the **declare-event** macro to define event slot symbols and to map these symbols to specific event data items returned by the X server (see paragraph 12.12.8, Declaring Event Types).

The following paragraphs describe each event type, listing its **event-key** keyword symbol and its slot name symbols. An event keyword symbol identifies a specific event type. An event keyword symbol can be given as an argument to **send-event** or to an event handler function; it can also appear in the *event-match* form of an **event-case** clause. An event slot name symbol identifies a specific event data item. Event slot names appear as keywords with associated values among the arguments passed to **send-event** or to an event handler function; as non-keyword symbols, they can also be in the *event-slots* form of an **event-case** clause.

In certain cases, more than one name symbol is defined for the same event slot. For example, in **:key-press** events, the symbols *window* and *event-window* both refer to the same event data item.

- **:invalid-time** if the specified time is earlier than the last-keyboard-grab time or later than the current server time. Otherwise, **grab-keyboard** returns a status of **:success** and last-keyboard-grab time is set to the specified time, with current-time replaced by current server time.

window — A **window**.

:owner-p — If true, all client windows receive keyboard input normally.

:sync-pointer-p — Indicates whether the pointer is in synchronous or asynchronous mode.

:sync-keyboard-p — Indicates whether the keyboard is in synchronous or asynchronous mode.

:time — A **timestamp**.

ungrab-keyboard *display* &key **:time** Function

Releases the keyboard if this client has it actively grabbed (from either **grab-keyboard** or **grab-key**), and releases any queued events. The request has no effect if the specified time is earlier than the last-keyboard-grab time or is later than the current server time. An **ungrab-keyboard** is performed automatically if the event window for an active keyboard grab becomes not viewable.

display — A **display**.

:time — A **timestamp**.

Grabbing a Key

12.11 The following paragraphs describe the functions used for passively grabbing and releasing the keyboard.

grab-key *window key* &key (**:modifiers** 0) **:owner-p :sync-pointer-p** **:sync-keyboard-p :time** Function

This request establishes a passive grab on the keyboard. If the specified *key* (which can also be a modifier key) is pressed (whether or not any specified modifier keys are down), and either of the following is true:

- *window* is an ancestor of (or is) the focus window
- *window* is a descendant of the focus window and contains the pointer
- These constraints are not satisfied for any ancestor of *window*, then the following occurs:
  - The keyboard is actively grabbed as described in **grab-keyboard**
  - The last-keyboard-grab time is set to the time that the *key* was pressed (as transmitted in the **:key-press** event)
  - The **:key-press** event is reported

The interpretation of the remaining arguments is as for **grab-keyboard**. The active grab is terminated automatically when the specified *key* has been released, independent of the state of the modifier keys.

ungrab-button *window button* &key (**:modifiers** 0) Function

Releases the passive button/key combination on the specified *window* if it was grabbed by this client. A zero *modifier* mask is equivalent to issuing the request for all possible modifier combinations including the combination of no modifiers. A *button* of **:any** is equivalent to issuing the request for all possible buttons. This has no effect on an active grab.

window — A **window**.

button — The button (type **card8**) that is released or **:any**.

:modifiers — A **modifier-mask**.

Grabbing the Keyboard

12.10 CLX provides the **grab-keyboard** and **ungrab-keyboard** functions for actively grabbing and releasing control of the keyboard.

grab-keyboard *window* &key **:owner-p :sync-pointer-p :sync-keyboard-p :time** Function

Returns:

grab-status — One of **:already-grabbed**, **:frozen**, **:invalid-time**, **:not-viewable**, or **:success**.

Actively grabs control of the keyboard. Further key events are reported only to the grabbing client. The request overrides any active keyboard grab by this client. **grab-keyboard** generates **:focus-in** and **:focus-out** events.

If **:owner-p** is **nil**, all generated key events are reported with respect to *window*. If **:owner-p** is true, then a generated key event that would normally be reported to this client is reported normally; otherwise the event is reported with respect to the *window*. Both **:key-press** and **:key-release** events are always reported, independent of any event selection made by the client.

If **:sync-keyboard-p** is **nil**, keyboard event processing continues normally (asynchronously); if the keyboard is currently frozen by this client, then processing of keyboard events is resumed. If **:sync-keyboard-p** is true, the keyboard (as seen via the protocol) appears to freeze, and no further keyboard events are generated by the server until the grabbing client issues a releasing **allow-events** request. Actual keyboard changes are not lost while the keyboard is frozen; they are simply queued for later processing.

If **:sync-pointer-p** is **nil**, pointer event processing is unaffected by activation of the grab. If **:sync-pointer-p** is true, the pointer (as seen via the protocol) appears to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing **allow-events** request. Actual pointer changes are not lost while the pointer is frozen; they are simply queued for later processing.

The grab can fail with a status of:

- **:already-grabbed** if the keyboard is actively grabbed by some other client
- **:frozen** if the keyboard is frozen by an active grab from another client
- **:not-viewable** if *window* is not viewable

Grabbing a Button

12.9 CLX provides the **grab-button** and **ungrab-button** functions for passively grabbing and releasing pointer control.

grab-button *window button event-mask* &key (**:modifiers** 0) **:owner-p** **:sync-pointer-p** **:sync-keyboard-p** **:confine-to** **:cursor** Function

This request establishes a passive grab. If the specified *button* is pressed when the specified modifier keys are down (and no other buttons or modifier keys are down), and:

- *window* contains the pointer
- The **:confine-to** window (if any) is viewable
- These constraints are not satisfied for any ancestor of *window*

then:

- The pointer is actively grabbed as described with **grab-pointer**
- The last-pointer-grab time is set to the time that the button was pressed (as transmitted in the **:button-press** event)
- The **:button-press** event is reported

The interpretation of the remaining arguments is the same as with **grab-pointer**. The active grab is terminated automatically when all buttons are released (independent of the state of modifier keys).

A zero *modifier* mask is equivalent to issuing the request for all possible modifier-key combinations (including the combination of no modifiers). It is not required that all specified modifiers have currently assigned keycodes. A *button* of **:any** is equivalent to issuing the request for all possible buttons. Otherwise, it is not required that the specified *button* currently be assigned to a physical button.

window — A **window**.

button — The button (type **card8**) pressed or **:any**.

event-mask — A **pointer-event-mask**.

:modifiers — A **modifier-mask**.

:owner-p — If true, all client windows receive pointer events normally.

:sync-pointer-p — Indicates whether the pointer is handled in a synchronous or asynchronous fashion.

:sync-keyboard-p — Indicates whether the keyboard is in synchronous or asynchronous mode.

:confine-to — A **window** to which the pointer is confined.

:cursor — A **cursor**.

window — The **window** grabbing the pointer.

event-mask — A **pointer-event-mask**.

:owner-p — If true, all client windows receive pointer events normally.

:sync-pointer-p — Indicates whether the pointer is in synchronous or asynchronous mode.

:sync-keyboard-p — Indicates whether the keyboard is in synchronous or asynchronous mode.

:confine-to — A **window** to which the pointer is confined.

:cursor — A **cursor**.

:time — A **timestamp**. A **nil** value means the current server time is used.

ungrab-pointer *display* &key **:time** Function

Releases the pointer if this client has it actively grabbed (from either **grab-pointer**, **grab-button**, or from a normal button press), and releases any queued events. The request has no effect if the specified **:time** is earlier than the last-pointer-grab time or is later than the current server time. An **ungrabpointer** is performed automatically if the event window or **:confine-to** window for an active pointer grab becomes not viewable.

This request generates **:enter-notify** and **:leave-notify** events.

display — A **display**.

:time — A **timestamp**.

change-active-pointer-grab *display event-mask* &optional *cursor time* Function

Changes the specified dynamic parameters if the pointer is actively grabbed by the client and the specified *time* is no earlier than the last-pointer-grab time and no later than the current server time. The interpretation of *event-mask* and *cursor* are as in **grab-pointer**. **change-active-pointer-grab** has no effect on the passive parameters of a **grab-button**.

display — A **display**.

event-mask — A **pointer-event-mask**.

cursor — A **cursor** or **nil**.

time — A **timestamp**.

Grabbing the Pointer

12.8 CLX provides the **grab-pointer** and **ungrab-pointer** functions for grabbing and releasing pointer control.

grab-pointer *window event-mask* &key **:owner-p :sync-pointer-p** Function
:sync-keyboard-p :confine-to :cursor :time
Returns:
grab-status — One of **:already-grabbed**, **:frozen**, **:invalid-time**, **:not-viewable**, or **:success**.

Actively grabs control of the pointer. Further pointer events are only reported to the grabbing client. The request overrides any active pointer grab by this client.

If **:owner-p** is **nil**, all generated pointer events are reported with respect to *window*, and are only reported if selected by *event-mask*. If **:owner-p** is true, and if a generated pointer event would normally be reported to this client, it is reported normally; otherwise the event is reported with respect to the *window*, and is only reported if selected by *event-mask*. For either value of **:owner-p**, unreported events are simply discarded.

If **:sync-pointer-p** is **nil**, pointer event processing continues normally (asynchronously); if the pointer is currently frozen by this client, then processing of pointer events is resumed. If **:sync-pointer-p** is true (indicating a synchronous action), the pointer (as seen via the protocol) appears to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing **allow-events** request. Actual pointer changes are not lost while the pointer is frozen; they are simply queued for later processing.

If **:sync-keyboard-p** is **nil**, keyboard event processing is unaffected by activation of the grab. If **:sync-keyboard-p** is true, the keyboard (as seen via the protocol) appears to freeze, and no further keyboard events are generated by the server until the grabbing client issues a releasing **allow-events** request. Actual keyboard changes are not lost while the keyboard is frozen; they are simply queued for later processing.

If **:cursor** is specified, it is displayed regardless of what window the pointer is in. Otherwise, the normal cursor for the *window* is displayed.

If a **:confine-to** window is specified, the pointer is restricted to stay within that window. The :**confine-to** window does not need to have any relationship to the *window*. If the pointer is not initially in the **:confine-to** window, it is warped automatically to the closest edge (with **:enter/:leave-events** generated normally) just before the grab activates. If the **:confine-to** window is subsequently reconfigured, the pointer is warped automatically as necessary to keep it contained in the window.

grab-pointer generates **:enter-notify** and **:leave-notify** events. **grab-pointer** can fail with a status of:

- **:already-grabbed** if the pointer is actively grabbed by some other client
- **:frozen** if the pointer is frozen by an active grab of another client
- **:not-viewable** if the *window* or the **:confine-to** window is not viewable, or if the **:confine-to** window lies completely outside the boundaries of the root window.
- **:invalid-time** if the specified time is earlier than the last-pointer-grab time or later than the current server time. Otherwise, the last-pointer-grab time is set to the specified time, with current-time replaced by the current server time, and a value of **:success** is returned by **grab-pointer**.

The specified *focus* window must be viewable at the time of the request. If the *focus* window later becomes not viewable, the new focus window depends on the *revert-to* argument. If *revert-to* is specified as **:parent**, the *focus* reverts to the parent (or the closest viewable ancestor) and the new *revert-to* value is take to be **:none**. If *revert-to* is **:pointer-root** or **:none**, the *focus* reverts to that value. When the *focus* reverts, **:focus-in** and **:focus-out** events are generated, but the last-focus-change time is not affected.

display — A **display**.

focus — The new input focus **window**.

revert-to — The focus **window** when focus is no longer viewable.

time — A **timestamp**.

input-focus *display* Function

Returns:
focus — Type (**or window** (**member :none :pointer-root**)).
revert-to — Type (**or window** (**member :none :pointer-root :parent**)).

Returns the *focus* window, **:pointer-root**, or **:none**, depending on the current state of the focus window. *revert-to* returns the current focus revert-to state.

display — A **display**.

warp-pointer-if-inside generates events just as if the user had instantaneously moved the pointer from one position to another. **warp-pointer-if-inside** cannot move the pointer outside the confine-to window of an active pointer grab; an attempt to do so only moves the pointer as far as the closest edge of the confine-to window.

destination — The **window** into which the pointer is moved.

destination-x, destination-y — The new position of the pointer relative to the *destination*.

source — The **window** that must currently contain the pointer.

source-x, source-y, source-width, source-height —The source rectangle that must currently contain the pointer.

warp-pointer-relative-if-inside *x-offset y-offset source source-x source-y* &optional (*source-width* 0) (*source-height* 0) Function

Moves the pointer by the given offsets. However, the move can only take place if the pointer is currently contained in a visible portion of the specified rectangle of the *source* window. If *source-height* is zero, it is replaced with the current height of *source-window* minus *source-y*. If *source-width* is zero, it is replaced with the current width of *source-window* minus *source-x*.

warp-pointer-relative-if-inside generates events just as if the user had instantaneously moved the pointer from one position to another. **warp-pointer-relative-if-inside** cannot move the pointer outside the confine-to window of an active pointer grab; an attempt to do so only moves the pointer as far as the closest edge of the confine-to window.

x-offset, y-offset — The offsets used to adjust the pointer position.

source — The **window** that must currently contain the pointer.

source-x, source-y, source-width, source-height — The source rectangle that must currently contain the pointer.

Managing Input Focus

12.7 CLX provides the **set-focus-input** and **focus-input** functions to set and get the keyboard input focus window.

set-input-focus *display focus revert-to* &optional *time* Function

Changes the keyboard input focus and the last-focus-change time. The function has no effect if the specified *time* is earlier than the current last-focus-change time or is later than the current server time; otherwise, the last-focus-change time is set to the specified *time*. The **set-input-focus** function causes the X server to generate **:focus-in** and **:focus-out** events.

If **:none** is specified as the *focus*, all keyboard events are discarded until a new focus window is set. In this case, the *revert-to* argument is ignored.

If a window is specified as the *focus* argument, it becomes the keyboard's focus window. If a generated keyboard event would normally be reported to this window or one of its inferiors, the event is reported normally; otherwise, the event is reported with respect to the focus window.

If **:pointer-root** is specified as the *focus* argument, the input focus window is set to the root window of the screen containing the pointer when each keyboard event occurs. In this case, the *revert-to* argument is ignored.

motion-events w*indow* &key **:start :stop** (**:result-type 'list**) Function

Returns:

motion-events — Type **(repeat-seq (int16 x) (int16 y) (timestamp time))**.

Many X server implementations maintain a more precise history of pointer motion between event notifications. The pointer position at each pointer hardware interrupt can be stored into a buffer for later retrieval.This is called the *motion history buffer*. A paint program, for example, may want to have a precise history of where the pointer traveled, even though for most other applications this amount of detail is grossly excessive.

The **motion-events** function returns all events in the motion history buffer that fall between the specified **:start** and **:stop** timestamps (inclusive) and have coordinates that lie within the specified *window* (including borders) at its present placement. If the **:start** time is later than the **:stop** time or if the **:start** time is in the future, no events are returned.

window — The **window** containing the returned motion events.

:start, :stop — **timestamp** values for the time interval for returned motion events.

:result-type — The form of the returned motion events.

warp-pointer *destination destination-x destination-y* Function

Moves the pointer to the given coordinates relative to the *destination* window. **warp-pointer** should be rarely be used since the user should normally be in control of the pointer position. **warp-pointer** generates events just as if the user had instantaneously moved the pointer from one position to another.

warp-pointer cannot move the pointer outside the confine-to window of an active pointer grab; an attempt to do so only moves the pointer as far as the closest edge of the confine-to window.

destination — The **window** into which the pointer is moved.

destination-x, destination-y — The new position of the pointer relative to the destination.

warp-pointer-relative *display x-offset y-offset* Function

Moves the pointer by the given offsets. This function should rarely be used since the user should normally be in control of the pointer position. **warp-pointer-relative** generates events just as if the user had instantaneously moved the pointer from one position to another.

warp-pointer-relative cannot move the pointer outside the confine-to window of an active pointer grab; an attempt to do so only moves the pointer as far as the closest edge of the confine-to window.

display — A **display**.

x-offset, y-offset — The offsets used to adjust the pointer position.

warp-pointer-if-inside *destination destination-x destination-y* Function
source source-x source-y &optional (*source-width* 0)
(*source-height* 0)

Moves the pointer to the given position relative to the *destination* window. However, the move can only take place if the pointer is currently contained in a visible portion of the specified rectangle of the *source* window. If *source-height* is zero, it is replaced with the current height of *source* window minus *source-y*. If *source-width* is zero, it is replaced with the current width of *source* window minus *source-x*.

event-mask — Specifies the event types that receiving clients must select.

event-slots — Keyword-value pairs that describe the contents of an event.

:propagate-p — If true, the event can be propagated to ancestors of the destination window.

:display — A **display**.

Pointer Position

12.6 The CLX functions affecting pointer position are discussed in the following paragraphs.

query-pointer *window* Function

Returns:
x — Type **int16**.
y — Type **int16**.
same-screen-p — Type **boolean**.
child — Type **window** or **null**.
state-mask — Type **card16**.
root-x — Type **int16**.
root-y — Type **int16**.
root — Type **window**.

Returns the current pointer coordinates relative to the given *window*. If **query-pointer** returns **nil** for *same-screen-p*, the pointer is not on the same screen as the *window*. In this case, **query-pointer** returns a value of **nil** for *child* and a value of zero for *x* and *y*. If **query-pointer** returns true for *same-screen-p*, the returned *x* and *y* are relative to the origin of window. The *child* is the child of the window containing the pointer, if any. The *state-mask* returned gives the current state of the modifier keys and pointer buttons. The returned *root* is the root window currently containing the pointer. The returned *root-x* and *root-y* specify the pointer coordinates relative to *root*.

window — A **window** specifying the coordinate system for the returned position.

global-pointer-position *display* Function

Returns:
root-x — Type **int16**.
root-y — Type **int16**.
root — Type **window**.

Returns the *root* window currently containing the *display* pointer and the current position of the pointer relative to the *root*.

display — A **display**.

pointer-position *window* Function

Returns:
x — Type **int16**.
y — Type **int16**.
same-screen-p — Type **boolean**.
child — Type **window** or **null**.

Returns the current pointer coordinates relative to the given *window*. If **pointer-position** returns **nil** for *same-screen-p*, the pointer is not on the same screen as the *window*. In this case, **pointer-position** returns a value of **nil** for *child* and a value of zero for *x* and *y*. If **pointer-position** returns true for *same-screen-p*, the returned *x* and *y* are relative to the origin of *window*.

window — A **window** specifying the coordinate system for the returned position.

event-listen *display* &optional (**timeout** 0) Function

Returns:
event-count — Type **(or null integer)**.

Returns the number of events queued locally. If the event queue is empty, **event-listen** waits for an event to arrive. If timeout is non-**nil** and no event arrives within the specified timeout interval (given in seconds), **event-listen** returns **nil**; if timeout is **nil**, **event-listen** will not return until an event arrives.

display — A **display**.

timeout — The number of seconds to wait for events.

with-event-queue *display* &body *body* Macro

Executes the *body* in a critical region in which the executing client process has exclusive access to the event queue.

display — A **display**.

body — Forms to execute.

Sending Events

12.5 A client can send an event to a window. Clients selecting this window event will receive it just like any other event sent by the X server.

send-event *window event-key event-mask* &rest *event-slots* &key Function
:propagate-p :display &allow-other-keys

Sends an event specified by the *event-key* and *event-slots* to the given destination *window*. Any active grabs are ignored. The *event-slots* passed depend on the event type. The keyword symbols used for each event type are event slot names defined by the **declare-event** macro and are described in paragraph 12.12.8, Declaring Event Types.

If the *window* is **:pointer-window**, the destination *window* is replaced with the window containing the pointer. If the *window* is **:input-focus**, the destination *window* is replaced with the descendant of the focus window that contains the pointer or (if no such descendant exists) the focus window. The **:display** keyword is only required if the *window* is **:pointer-window** or **:input-focus**.

The *event-key* must be one of the core events, or one of the events defined by an extension, so the server can send the event with the correct byte-order. The contents of the event are otherwise unaltered and unchecked by the server, except that the **send-event-p** event slot is set to true.

If the *event-mask* is **nil**, the event is sent to the client that created the destination *window* with an *event-mask* of 0; if that client no longer exists, no event is sent. Otherwise, the event is sent to every client selecting any of the event types specified by *event-mask* on the destination *window*.

If **:propagate-p** is true and no clients have selected any of the event types in *event-mask* on the destination *window*, the destination is replaced with the closest ancestor of *window* for which some client has selected a type in *event-mask* and no intervening window has that type in its do-not-propagate mask. If no such window exists, or if the *window* is an ancestor of the focus window and **:input-focus** was originally specified as the destination, the event is not sent to any clients. Otherwise, the event is reported to every client selecting on the final destination any of the types specified in *event-mask*.

window — The destination **window** for the event.

event-key — An **event-key** defining the type of event to send.

When a *test-form* returns true and **:peek-p** is **nil**, or when a *test-form* returns **nil** and **:discard-p** is true, the matching event is removed from the event queue before the body *forms* are executed.

display — A **display**.

:handler — A handler function or a sequence of handler functions.

:timeout — Specifies the timeout delay in seconds.

:peek-p — If **nil**, events are removed from the event queue after processing.

:discard-p — If true, unprocessed events are discarded.

:force-output-p — If true, buffered output requests are sent.

clauses — Code to process specified event types.

Managing the Event Queue

12.4 The following paragraphs describe CLX functions and macros used to:

- Put a new event on the event queue
- Discard the current event
- Return the current length of the event queue
- Gain exclusive access to the event queue for a client process

queue-event *display event-key* &rest *event-slots* &key **:append-p** &allow-other-keys — Function

Places an event of the type given by *event-key* into the event queue. When **:append-p** is true, the event is placed at the tail of the queue; otherwise, the event is placed at the head of the queue. The actual *event-slots* passed depend on the event type. The keyword symbols used for each event type are event slot names defined by the **declare-event** macro and are described in paragraph 12.12.8, Declaring Event Types.

display — A **display**.

event-key — Specifies the type of event placed in the queue.

event-slots — Keyword-value pairs that describe the contents of an event.

:append-p — If true, the event is placed at the tail of the queue; otherwise, the event is placed at the head of the queue.

discard-current-event *display* — Function

Returns:
discarded-p — Type **boolean**.

Discards the current event for the *display*. Returns **nil** when the event queue is empty; otherwise, returns **t**. This function provides extra flexibility for discarding events, but it should be used carefully; use **event-cond** instead, if possible. Typically, **discard-current-event** is called inside a handler function or a clause of an **event-case** form and is followed by another call to **process-event**, **event-case**, or **event-cond**.

display — A **display**.

event-case *display* &key **:timeout :peek-p :discard-p (:force-output-p t)** Macro
&body *clauses*
Returns:
handled-p — Type **boolean**.

Executes the matching clause for each queued event until a clause returns non-**nil**. The non-**nil** clause value is then returned. Each of the clauses is a list of the form (*event-match* [*event-slots*] &rest *forms*), where:

event-match — Either an **event-key**, a list of **event-keys**, otherwise, or **t**. It is an error for the same key to appear in more than one clause.

event-slots — If given, a list of (non-keyword) event slot symbols defined for the specified event type(s). See paragraph 12.12.8, Declaring Event Types.

forms — A list of forms that process the specified event type(s). The value of the last form is the value returned by the clause.

A clause matches an event if the **event-key** is equal to or a member of the *event-match*, or if the *event-match* is **t** or **otherwise**. If no **t** or **otherwise** clause appears, it is equivalent to having a final clause that returns **nil**. If *event-slots* is given, these symbols are bound to the value of the corresponding event slot in the clause forms. Each element of *event-slots* can also be a list of the form (*event-slot-keyword variable*), in which case the *variable* symbol is bound to the value of the event slot specified by the *event-slot-keyword*.

If every clause returns **nil** for each event in the event queue, **event-case** waits for another event to arrive. If **:timeout** is non-**nil** and no event arrives within the specified timeout interval (given in seconds), **event-case** returns **nil**; if **:timeout** is **nil**, **event-case** will not return until a clause returns non-**nil**. **event-case** may wait only once on network data and therefore timeout prematurely.

If **:force-output-p** is true, **event-case** first invokes **display-force-output** to send any buffered requests. If **:peek-p** is true, a processed event is not removed from the queue. If **:discard-p** is true, unprocessed events are removed from the queue; otherwise, unprocessed events are left in place.

display — A **display**.

:handler — A handler function or a sequence of handler functions.

:timeout — Specifies the timeout delay, in seconds.

:peek-p — If **nil**, events are removed from the event queue after processing.

:discard-p — If true, unprocessed events are discarded.

:force-output-p — If true, buffered output requests are sent.

clauses — Code to process specified event types.

event-cond *display* &key **:timeout :peek-p :discard-p (:force-output-p t)** Macro
&body *clauses*
Returns:
handled-p — Type **boolean**.

Similar to **event-case** except that each of the clauses is a list of the form (*event-match* [*event-slots*] *test-form* &rest *forms*). Executes the *test-form* of the clause that matches each queued event until a *test-form* returns non-**nil**. The body *forms* of the clause are then executed. The values returned by the last clause body form are then returned by **event-cond**.

Processing Events

12.3 Events received by a CLX client are stored in an *event queue* until they are read and processed. Events are processed by *handler functions*.

handler-function &rest *event-slots* &key **:display :event-key :send-event-p** &allow-other-keys — Function

Returns:
handled-p — Type **boolean**.

The arguments to a handler function are keyword-value pairs that describe the contents of an event. The actual *event-slots* passed depend on the event type, except that **:display**, **:event-key**, and **:send-event-p** are given for all event types. The keyword symbols used for each event type are event slot names defined by the **declare-event** macro and are described in paragraph 12.12.8, Declaring Event Types.

If a handler returns non-**nil**, the event is considered *processed* and can be removed from the event queue. Otherwise, if a handler function returns **nil**, the event can remain in the event queue for later processing.

:display — A **display** for the connection that returned the event.

:event-key — An **event-key** keyword specifying the event type.

:send-event-p — If true, the event was sent from another application using the **send-event** function.

process-event *display* &key **:handler :timeout :peek-p :discard-p (:force-output-p t)** — Function

Returns:
handled-p — Type **boolean**.

Invokes **:handler** on each queued event until **:handler** returns non-**nil**. Then, the non-**nil :handler** value is returned by **process-event**. If **:handler** returns **nil** for each event in the event queue, **process-event** waits for another event to arrive. If timeout is non-**nil** and no event arrives within the specified timeout interval (given in seconds), **process-event** returns **nil**; if timeout is **nil**, **process-event** will not return until **:handler** returns non-**nil**. **process-event** may wait only once on network data, and therefore timeout prematurely.

If **:force-output-p** is true, **process-event** first invokes **display-force-output** to send any buffered requests. If **:peek-p** is true, a processed event is not removed from the queue. If **:discard-p** is true, unprocessed events are removed from the queue; otherwise, unprocessed events are left in place.

If **:handler** is a sequence, it is expected to contain handler functions for each event type. The sequence index of the handler function for a particular event type is given by (**position event-key *event-key-vector***).

display — A **display**.

:handler — A handler function or a sequence of handler functions.

:timeout — Specifies the timeout delay in seconds.

:peek-p — If **nil**, events are removed from the event queue after processing.

:discard-p — If true, unprocessed events are discarded.

:force-output-p — If true, buffered output requests are sent.

Event Mask Keyword	Event Types Selected
:button-1-motion	**:motion-notify** when **:button-1** is down
:button-2-motion	**:motion-notify** when **:button-2** is down
:button-3-motion	**:motion-notify** when **:button-3** is down
:button-4-motion	**:motion-notify** when **:button-4** is down
:button-5-motion	**:motion-notify** when **:button-5** is down
:button-motion	**:motion-notify** when any pointer button is down
:button-press	**:button-press**
:button-release	**:button-release**
:colormap-change	**:colormap-notify**
:enter-window	**:enter-notify**
:exposure	**:exposure**
:focus-change	**:focus-in, :focus-out**
:key-press	**:key-press**
:key-release	**:key-release**
:keymap-state	**:keymap-notify**
:leave-window	**:leave-notify**
:owner-grab-button	Pointer events while button is grabbed
:pointer-motion	**:motion-notify**
:pointer-motion-hint	Single **:motion-notify** only
:property-change	**:property-notify**
:resize-redirect	**:resize-request**
:structure-notify	**:circulate-notify, :configure-notify, :destroy-notify, : gravity-notify, :map-notify, :reparent-notify, :unmap-notify**
:substructure-redirect	**:circulate-request, :configure-request, :map-request**
:visibility-change	**:visibility-notify**

Some types of events do not have to be selected to be received and therefore are not represented in an event-mask. For example, the **copy-plane** and **copy-area** functions cause **:graphics-exposure** and **:no-exposure** events to be reported, unless exposures are turned **:off** in the graphics context (see **copy-area** and **copy-plane** in paragraph 6.2, Area and Plane Operations, and **gcontext-exposures** in paragraph 5.4.6, Exposures). Also, **:selection-clear**, **:selection-request**, **:selection-notify** and **:client-message** events can be received at any time, but they are generally sent only to clients using selections (see paragraph 12.12.7, Client Communications Events). **:mapping-notify** is always sent to clients when the keyboard mapping is changed.

Any client can select events for any window. A window maintains a separate event-mask for each interested client. In general, multiple clients can select for the same events on a window. After the X server generates an event, it sends it to all clients which selected it. However, the following restrictions apply to sharing window events among multiple clients. For a given window:

- Only one client at a time can include **:substructure-redirect** in its event-mask
- Only one client at a time can can include **:button-press** in its event-mask
- Only one client at a time can include **:resize-redirect** in its event-mask

Client programs can override the server's normal distribution of events by *grabbing* the pointer or the keyboard. Grabbing causes events from the pointer or keyboard device to be reported to a single specified window, rather than to their ordinary destinations. It can also cause the server to *freeze* the grabbed device, sending queued events only when explicitly requested by the grabbing client. Two kinds of grabs are possible:

- Active — Events are immediately grabbed.
- Passive — Events are grabbed later, as soon as a specified device event occurs.

Grabbing an input device is performed rarely and usually only by special clients, such as window managers.

This section describes the CLX functions used to:

- Select events
- Process an event on the event queue
- Manage the event queue
- Send events to other applications
- Read and change the pointer position
- Manage the keyboard input focus
- Grab pointer and keyboard events
- Release queued events

This section also contains a detailed description of the content of each type of event.

Selecting Events

12.2 A client *selects* which types of events it receives from a specific window. The window event-mask attribute, set by the client, determines which event types are selected (see **window-event-mask** in paragraph 4.3, Window Attributes). Most types of events are received by a client only if they are selected for some window.

In the X protocol, an event-mask is represented as a bit string. CLX also allows an event mask to be defined by a list of **event-mask-class** keywords. The functions **make-event-keys** and **make-event-mask** can be used to convert between these two forms of an event-mask. In general, including an **event-mask-class** keyword in an event-mask causes one or more related event types to be selected. The following table describes the event types selected by each **event-mask-class** keyword.

12 EVENTS AND INPUT

Introduction

12.1 A client application uses CLX functions to send *requests* to an X server over a display connection returned by the **open-display** function. In return, the X server sends back *replies* and *events*. Replies are synchronized with specific requests and return requested server information. Events typically occur asynchronously. Device events are generated by user input from both the keyboard and pointer devices. Other events are side-effects of the requests sent by CLX functions. The types of events returned by an X server are summarized below.

Device Events	Events Returned
Keyboard	**:key-press** **:key-release**
Pointer	**:button-press** **:button-release** **:enter-notify** **:leave-notify** **:motion-notify**

Side-Effect Events	Events Returned
Client communication	**:client-message** **:property-notify** **:selection-clear** **:selection-notify** **:selection-request**
Color map state	**:colormap-notify**
Exposure	**:exposure** **:graphics-exposure** **:no-exposure**
Input focus	**:focus-in** **:focus-out**
Keyboard and pointer state	**:keymap-notify** **:mapping-notify**
Structure control	**:circulate-request** **:configure-request** **:map-request** **:resize-request**
Window state	**:circulate-notify** **:configure-notify** **:create-notify** **:destroy-notify** **:gravity-notify** **:map-notify** **:reparent-notify** **:unmap-notify** **:visibility-notify**