selection-owner *display selection* &optional *time* Function

Returns:

owner — Type **window** or **null**.

Returns and (with **setf**) changes the owner and the last-changed *time* for the *selection*. If the owner is **nil**, no owner for the *selection* exists. When the owner window for a *selection* is destroyed, the *selection* owner is set to **nil** without affecting the last-changed *time*.

The *time* argument is used only when changing the *selection* owner. If the *time* is **nil**, the current server time is used. If the *time* is earlier than the current last-changed time of the *selection* or if the *time* is later than the current server time, the owner is not changed. Therefore, a client should always confirm successful change of ownership by immediately calling **selection-owner**. If the change in ownership is successful, the last-changed time of the *selection* is set to the specified *time*.

If the change in ownership is successful and the new owner is different from the previous owner, and if the previous owner is not **nil**, a **:selection-clear** event is generated for the previous owner window.

NOTE: Standard conventions for inter-client communication require that a non-nil time must be specified. If possible, the time should be the time of a user event which initiated the change of ownership. Alternatively, a timestamp can be obtained by calling change-property to append zero-length data to some property; the timestamp in the resulting **:property-notify** event can then be used.

---

display — A **display**.

selection — The **xatom** for the selection name.

time — A **timestamp**.

When multiple clients negotiate for ownership of a selection, certain race conditions might be possible. For example, two clients might each receive a user command to assert ownership of the **:primary** selection, but the order in which the server processes these client requests is unpredictable. As a result, the ownership request initiated most recently by the user might be incorrectly overridden by the other earlier ownership request. To prevent such anomalies, the server records a *last-changed* timestamp for each change of selection ownership.

Although inter-client communication via selections is rather complex, it offers important benefits. Since selection communication is mediated by an X server, clients can share data even though they are running on different hosts and using different networking protocols. Data storage and conversion is distributed among clients so that the server is not required to provide all possible data types or to store multiple forms of selection data.

Certain predefined atoms are used as standard selections, as described in the X11 Interclient Communications Conventions Manual. Some of the standard selections covered by these conventions are:

- **:primary** — The *primary selection*. The main vehicle for inter-client cut and paste operations.
- **:secondary** — The *secondary selection*. In some environments, clients can use this as an auxiliary to **:primary**.
- **:clipboard** — Analogous to akill ring. Represents the most recently deleted data item.

convert-selection *selection type requestor* &optional *property time* Function

Requests that the value of the *selection* be converted to the specified *type* and stored in the given *property* of the *requestor* window.

If the *selection* has an owner, the X server sends a :**selection-request** event to the owner window. Otherwise, if no owner exists, the server generates on the requestor a **:selection-notify** event containing a **nil** *property* atom.

The given *property* specifies the requestor property that will receive the converted value. If the *property* is omitted, the *selection* owner will define a property to use. The *time* furnishes a timestamp representing the time of the conversion request; by default, the current server time is used.

NOTE: Standard conventions for inter-client communication require that both the requestor property and the time must be specified. If possible, the time should be the time of a user event which initiated the conversion. Alternatively, a timestamp can be obtained by calling **change-property** to append zero-length data to some property; the timestamp in the resulting **:property-notify** event can then be used.

---

selection — The **xatom** for the selection name.

type — The **xatom** for the requested data type.

requestor — The **window** to receive the converted *selection* value.

property — The **xatom** for the requestor property to receive the converted value.

time — A **timestamp**.

If (**mod** *delta* (**length** *properties*)) is non-zero, a **:property-notify** event is generated on the window for each property, in the same order as they appear in the *properties* sequence.

window — A **window**.

properties — A sequence of **xatom** values.

delta — The index interval between source and destination elements of *properties*.

Selections

11.3 A selection is an atom used to identify data that can be shared among all client programs connected to an X server. Unlike properties, the data represented by a selection is stored by some client program, not by the server.

The data named by a selection is associated with a client window, which is referred to as the *selection owner*. The server always knows which window is the owner of a selection. Selections can be created freely by clients using **intern-atom** to create an atom. CLX provides functions to inquire or change the owner of a selection and to *convert* a selection.

Conversion is the key to the use of selections for inter-client communication. Suppose Client A wants to paste the contents of the data named by selection *S* into his window *WA*. Client A calls **convert-selection** on selection atom *S*, sending a conversion request to the server. The server, in turn, sends a **:selection-request** event to the current owner of *S*, which is window *WB* belonging to Client B. The **:selection-request** event contains the *requestor* window (*WA*), the selection atom (*S*), an atom identifying a requested data type, and the name of a property of *WA* into which the value of *S* will be stored.

Since *WB* is the owner of *S*, it must be associated with the data defined by Client B as the value of *S*. When *WB* gets the **:selection-request** event, Client B is expected to convert the value of *S* to the requested data type (if possible) and store the converted value in the given requestor property. Client B is then expected to send a **:selection-notify** event to the requestor window *WA*, informing the requestor that the converted value for *S* is ready. Upon receiving the **:selection-notify** event, Client A can call **get-property** to retrieve the converted value and to paste it into *WA*.

NOTE: Clients using selections must always be prepared to handle **:selection-request** events and/or **:selection-notify** events. There is no way for a client to ask not to receive these types of events.

Type atoms used in selection conversion can represent arbitrary client-defined interpretations of the selection data. For example, if the value of selection *S* is a text string, Client A might request its typeface by requesting conversion to the **:font** type. A type **atom** can also represent a request to the selection owner to perform some action as a side-effect of conversion (for example, **:delete**). Some of the predefined atoms of an X server are intended to be used as selection types (for example, **:colormap**, **:bitmap**, **:string**, and so forth) However, X does not impose any requirements on the interpretation of type atoms.

get-property *window property* &key **:type** (**:start** 0) **:end :delete-p** (**:result-type 'list**) **:transform** Function

Returns:
data — Type **sequence**.
type — Type **xatom**.
format — Type (**member 8 16 32**).
bytes-after — Type **card32**.

Returns a subsequence of the data for the window property. The **:start** and **:end** indexes specify the property *data* elements returned. The **:transform** function is called for elements of the specified subsequence to compute the *data* sequence returned. The property *type* and *format* are also returned. The final return value gives the actual number of data bytes (not elements) following the last data element returned.

If the *property* does not exist, the returned *data* and *type* are **nil** and the returned *format* and *bytes-after* are zero.

If the given **:type** is non-**nil** but does not match the actual property type, then the *data* returned is **nil**, the *type* and *format* returned give the actual property values, and the *bytes-after* returned gives the total number of bytes (not elements) in the property data.

If the given **:type** is **nil** or if it matches the actual property type, then:

- The *data* returned is the transformed subsequence of the property data.
- The *type* and *format* returned give the actual property values.
- The *bytes-after* returned gives the actual number of data bytes (not elements) following the last data element returned.

In this case, the **:delete-p** argument is also examined. If **:delete-p** is true and *bytes-after* is zero, the property is deleted and a **:property-notify** event is generated for the *window*.

window — A **window**.

property — A property name **xatom**.

:type — The requested type **xatom** or **nil**.

:start, **:end** — Specify the subsequence of property *data* returned.

:transform — A function that transforms each data element into a data value to return.

:delete-p — If true, the existing *property* can be deleted.

:result-type — The type of data sequence to return. Default is **'list**.

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

Returns:
properties — Type **sequence** of **keyword**.

Returns a sequence containing the names of all *window properties*.

window — A **window**.

:result-type — The type of sequence to return. Default is **'list**.

rotate-properties *window properties* &optional (*delta* 1) Function

Rotates the values of the given *window properties*. The value of property *i* in the given sequence is changed to the value of the property at index (**mod** (+ *i delta*) (**length** *properties*)). This function operates much like the **rotatef** macro in Common Lisp.

- Create or change a property
- Return property data
- List window properties
- Delete a property

change-property *window property data type format* &key (**:mode :replace**) (**:start** 0) **:end :transform** Function

Creates a new window property or changes an existing property. A **:property-notify** event is generated for the *window*.

If the **:mode** is **:replace**, the new *data*, *type*, and *format* replace any previous values. The subsequence of previous data elements that are replaced is defined by the **:start** and **:end** indexes.

If the **:mode** is **:prepend** or **:append**, no previous data is changed, but the new *data* is added at the beginning or the end, respectively. For these modes, if the *property* already exists, the new *type* and *format* must match the previous values.

The **:transform**, if given, is a function used to compute the actual property data stored. The **:transform**, which must accept a single data element and return a single transformed data element, is called for each data element. If the *data* is a string, the default **:transform** function transforms each character into its ASCII code; otherwise, the default is to store the *data* unchanged.

window — A **window**.

property — A property name **xatom**.

data — A sequence of property data elements.

type — The property type **xatom**.

format — One of 8, 16, or 32.

:mode — One of **:replace**, **:append**, or **:prepend**.

:start, :end — Specify the subsequence of previous data replaced when **:mode** is **:replace**.

:transform — A function that transforms each data element into a data value to store.

delete-property *window property* Function

Deletes the *window property*. If the *property* already exists, a **:property-notify** event is generated for the *window*.

window — A **window**.

property — A property name **xatom**.

When creating a new atom, the following conventions should be obeyed in order to minimize the conflict between atom names:

- Symbol names beginning with an underscore should be used for atoms that are private to a particular vendor or organization. An additional prefix should identify the organization.
- Symbol names beginning with two underscores should be used for atoms that are private to a single application or end user.

CLX provides functions to convert between an **xatom** and its corresponding ID integer. The data type of an atom ID is **card29**. The **xatom** representation is usually sufficient for most CLX programs. However, it is occasionally useful to be able to convert an atom ID returned in events or properties into its corresponding **xatom**.

atom-name *display atom-id* Function

Returns:
atom-name — Type **keyword**.

Returns the atom keyword for the *atom-id* on the given *display* server.

display — A **display** object.

atom-id — A **card29**.

find-atom *display atom-name* Function

Returns:
atom-id — Type **card29** or **null**.

Returns the atom ID for the given *atom-nam*e, if it exists. If no atom of that name exists for the display server, **nil** is returned.

display — A **display** object.

atom-name — An **xatom**.

intern-atom *display atom-name* Function

Returns:
atom-id — Type **card29** or **null**.

Creates an atom with the given name and returns its atom ID. The atom can survive the interning client; it exists until the last server connection has been closed and the server resets itself.

display — A **display** object.

atom-name — An **xatom**.

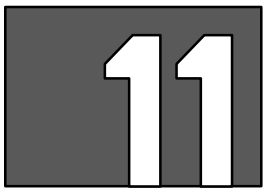
Properties

11.2 For each window, an X server can record a set of *properties*. Properties are a general mechanism for clients to associate arbitrary data with a window, and for clients to communicate window data to each other via the server. No interpretation is placed on property data by the server itself.

A property consists of a name, a type, a data format, and data. The name of a property is given by an atom. The property type is another atom used to denote the intended interpretation of the property data. The property formats specifies whether the property data should be treated as a set of 8-, 16-, or 32-bit elements. The property format must be specified so that the X server can communicate property data with the correct byte order.

CLX provides functions to:

ATOMS, PROPERTIES, AND SELECTIONS



Atoms

11.1 In X, an *atom* is a unique ID used as the name for certain server resources — properties and selections.

In CLX, an atom is represented by a keyword symbol. For convenience, CLX functions also allow atoms to be specified by strings and non-keyword symbols. **xatom** is a CLX data type that permits either string or symbol values. A string is equivalent to the **xatom** given by (**intern** *string* **'keyword**). A symbol is equivalent to the **xatom** given by (**intern** (**symbol-name** *symbol*) **'keyword**). The symbol name string of an **xatom** must consist only of ISO Latin characters. Note that the case of **xatom** strings is important; the **xatom** "Atom" is not the same as the **xatom** "ATOM".

Certain atoms are already predefined by every X server. Predefined atoms are designed to represent common names that are likely to be useful for many client applications. Note that these atoms are predefined only in the sense of having **xatom** and **card29** values, not in the sense of having required semantics. No interpretation is placed on the meaning or use of an atom by the server. The **xatom** objects predefined by CLX are listed below.

:arc
:atom
:bitmap
:cap_height
:cardinal
:colormap
:copyright
:cursor
:cut_buffer0
:cut_buffer1
:cut_buffer2
:cut_buffer3
:cut_buffer4
:cut_buffer5
:cut_buffer6
:cut_buffer7
:drawable
:end_space
:family_name
:font
:font_name
:full_name
:integer
:italic_angle
:max_space
:min_space
:norm_space
:notice
:pixmap
:point
:point_size
:primary
:quad_width
:rectangle
:resolution
:resource_manager
:rgb_best_map
:rgb_blue_map
:rgb_color_map
:rgb_default_map
:rgb_gray_map
:rgb_green_map
:rgb_red_map
:secondary
:strikeout_ascent
:strikeout_descent
:string
:subscript_x
:subscript_y
:superscript_x
:superscript_y
:underline_position
:underline_thickness
:visualid
:weight
:window
:wm_class
:wm_client_machine
:wm_command
:wm_hints
:wm_icon_name
:wm_icon_size
:wm_name
:wm_normal_hints
:wm_size_hints
:wm_transient_for
:wm_zoom_hints
:x_height