# Elk/Xlib Reference Manual

*Oliver Laumann*

## 1. Introduction

This manual lists the functions, special forms, and variables defined by the Xlib extension included in the Elk distribution. Most of the functions are directly equivalent to a function of the Xlib C library, so that the description need not be repeated. In such cases, only the name of the corresponding Xlib function is mentioned. Thus, you should have the *Xlib − C Language X Interface* manual within reach when using this reference manual.

The functions listed in this document can be loaded by evaluating the expression

```
(require 'xlib).
```

in the interpreter's top level or in a Scheme program.

The types of arguments of the procedures listed below are not described when they are obvious from the context or from the name. For instance, an argument named *window* is always of type *window*, an argument named *atom* is an object of type *atom*, etc. Arguments the names of which end in "?" are always of type *boolean*.

If a function returns several items of the same type (for instance, a list of windows), the return value is a vector of objects of this type. If a function returns a collection of items of different types or of different semantics, the return value is a list of objects (or a pair). In this case, *multiple-value-bind* can be used to bind variables to the return values.

In the following, each description of a procedure, special form, or variable lists the kind of object in boldface. Here, **procedure** denotes either a primitive procedure or a compound procedure, **syntax** denotes a special form or a macro, and **variable** denotes a global variable that has some initial value and can be re-assigned a new value by the user (by means of *set!* or *fluid-let*).

## 2. Display Functions

**(display?** *x***)** **procedure**

Returns #t iff *x* is an object of type *display*.

**(open-display** . *name-of-display***)** **procedure**

See *XOpenDisplay*. *name-of-display* is a string or a symbol. If no name is specified, a NULL name will be passed to *XOpenDisplay*.

**(close-display** *display***)** **procedure**

See *XCloseDisplay*. Finalizes all objects associated with the display, then closes the display.

**(display-default-root-window** *display***)** **procedure**
**(display-root-window** *display***)** **procedure**

See *XDefaultRootWindow*.

**(display-default-colormap** *display***)** **procedure**
**(display-colormap** *display***)** **procedure**

See *XDefaultColormap*. Returns the default colormap of the display's default screen.

**(display-default-gcontext** *display***)** **procedure**

See *XDefaultGC*. Returns the default graphics context of the display's default screen.

**(display-default-depth** *display***)** **procedure**

See *XDefaultDepth*. Returns the default depth of the display's default screen.

**(display-default-screen-number** *display***)** **procedure**

See *XDefaultScreen*. Returns an integer.

**(display-cells** *display screen-number***)** **procedure**

See *XDisplayCells*. Returns an integer.

**(display-planes** *display screen-number***)** **procedure**

See *XDisplayPlanes*. Returns an integer.

**(display-string** *display***)** **procedure**

See *XDisplayString*. Returns a string.

**(display-vendor** *display***)** **procedure**

See *XServerVendor*, *XVendorRelease*. Returns a pair; the car is a string (the vendor identification), and the cdr is an integer (the vendor release number).

**(display-protocol-version** *display***)** **procedure**

See *XProtocolVersion*, *XProtocolRevision*. Returns a pair of integers (the X protocol's major and minor version numbers).

**(display-screen-count** *display***)** **procedure**

See *XScreenCount*. Returns an integer.

**(display-image-byte-order** *display***)** **procedure**

See *XImageByteOrder*. Returns a symbol (`lsb-first` or `msb-first`).

**(display-bitmap-unit** *display***)** **procedure**

See *XBitmapUnit*. Returns an integer.

**(display-bitmap-bit-order** *display***)** **procedure**

See *XBitmapBitOrder*. Returns a symbol (`lsb-first` or `msb-first`).

**(display-bitmap-pad** *display***)** **procedure**

See *XBitmapPad*. Returns an integer.

**(display-width** *display***)** **procedure**
**(display-height** *display***)** **procedure**

See *XDisplayWidth*, *XDisplayHeight*. Returns the width/height of the display's default screen.

**(display-width-mm** *display***)** **procedure**
**(display-height-mm** *display***)** **procedure**

See *XDisplayWidthMM*, *XDisplayHeightMM*. Returns the width/height of the display's default screen in millimeters.

**(display-motion-buffer-size** *display***)** **procedure**

See *XDisplayMotionBufferSize*. Returns an integer.

**(display-flush-output** *display***)** **procedure**

See *XFlush*.

**(display-wait-output** *display discard-events?***)** **procedure**

See *XSync*.

**(no-op** *display***)** **procedure**

See *XNoOp*.

**(list-depths** *display screen-number***)** **procedure**

See *XListDepths*. Returns a vector of integers.

**(list-pixmap-formats** *display***)** **procedure**

See *XListPixmapFormats*. Returns a vector of lists of three integers (depth, bits per pixel, and scanline pad).

**(set-after-function!** *display procedure***)** **procedure**

See *XSetAfterFunction*. Returns the old after function. If *procedure* is #f, the current after function is disassociated from the display.

**(after-function** *display***)** **procedure**

Returns the after function currently associated with the given display (#f if there is none).

**(synchronize** *display***)** **procedure**

Sets the display's after function to *display-wait-output*.

## 3. Window Functions

**(window?** *x***)** **procedure**

Returns #t iff *x* is an object of type *window*.

**(drawable?** *x***)** **procedure**

Returns #t iff *x* is a "drawable" (window or pixmap).

**(window-display** *window***)** **procedure**

Returns the display associated with the window.

**(create-window** . *args***)** **procedure**

See *XCreateWindow*. This function is used to create a new window.

The number of arguments must be even. The 1st, 3rd, etc. argument is the name (a symbol) of an attribute to be set when the window is created, the 2nd, 4th, etc. argument is the corresponding value. The attributes can be specified in any order.

Attributes are *x*, *y*, *width*, *height*, *border* (each of which has an integer value), *parent* (the parent window), and all attributes that can be set by means of the `set-window-`*attribute*`!` functions below except *sibling* and *stack-mode*. The attributes *parent*, *width*, and *height* are mandatory. The default for *x* and *y* is 0, the default for *border* is 2.

**(set-window-x!** *window value***)** **procedure**
**(set-window-y!** *window value***)** **procedure**
**(set-window-width!** *window value***)** **procedure**
**(set-window-height!** *window value***)** **procedure**
**(set-window-border-width!** *window value***)** **procedure**
**(set-window-sibling!** *window value***)** **procedure**
**(set-window-stack-mode!** *window value***)** **procedure**
**(set-window-background-pixmap!** *window value***)** **procedure**
**(set-window-background-pixel!** *window value***)** **procedure**
**(set-window-border-pixmap!** *window value***)** **procedure**
**(set-window-border-pixel!** *window value***)** **procedure**

**(set-window-bit-gravity!** *window value***)** **procedure**
**(set-window-gravity!** *window value***)** **procedure**
**(set-window-backing-store!** *window value***)** **procedure**
**(set-window-backing-planes!** *window value***)** **procedure**
**(set-window-backing-pixel!** *window value***)** **procedure**
**(set-window-save-under!** *window value***)** **procedure**
**(set-window-event-mask!** *window value***)** **procedure**
**(set-window-do-not-propagate-mask!** *window value***)** **procedure**
**(set-window-override-redirect!** *window value***)** **procedure**
**(set-window-colormap!** *window value***)** **procedure**
**(set-window-cursor!** *window value***)** **procedure**

See *XConfigureWindow*, *XChangeWindowAttributes*. Set the sibling window, stacking mode, background pixmap, background pixel, border pixel, cursor, and other attributes (see the `window-` functions below) of the specified window.

The stacking mode is a symbol (`above`, `below`, `top-if`, `bottom-if`, `opposite`). The *value* argument to *set-window-sibling!* must be a window, *set-window-background-pixmap!* expects a pixmap, *set-window-background-pixel!* and *set-window-border-pixel!* expect a pixel, and *set-window-cursor!* expects a cursor argument. For the types of the *value* argument of the other functions see the return values of the *window-* functions below.

**(window-x** *window***)** **procedure**
**(window-y** *window***)** **procedure**
**(window-width** *window***)** **procedure**
**(window-height** *window***)** **procedure**
**(window-border-width** *window***)** **procedure**
**(window-depth** *window***)** **procedure**
**(window-visual** *window***)** **procedure**
**(window-root** *window***)** **procedure**
**(window-class** *window***)** **procedure**
**(window-bit-gravity** *window***)** **procedure**
**(window-gravity** *window***)** **procedure**
**(window-backing-store** *window***)** **procedure**
**(window-backing-planes** *window***)** **procedure**
**(window-backing-pixel** *window***)** **procedure**
**(window-save-under** *window***)** **procedure**
**(window-colormap** *window***)** **procedure**
**(window-map-installed** *window***)** **procedure**
**(window-map-state** *window***)** **procedure**
**(window-all-event-masks** *window***)** **procedure**
**(window-your-event-mask** *window***)** **procedure**
**(window-do-not-propagate-mask** *window***)** **procedure**
**(window-override-redirect** *window***)** **procedure**
**(window-screen** *window***)** **procedure**

See *XGetWindowAttributes*. Returns the x and y coordinates, width, height, border width, depth, visual, root window, class, bit gravity, window gravity, backing store availability, backing planes, backing pixel, save under availability, colormap, colormap installation information, map state, global event mask, local event mask, "do-not-propagate" mask, override redirect attribute, and screen of the specified window.

*window-visual* and *window-screen* always return the empty list in the current release of the software. *window-root* returns a window. *window-class* returns a symbol (`input-output`, `input-only`). *window-bit-gravity* returns a symbol (`forget`, `north-west`, `north`, `north-east`, `west`, `center`, `east`, `south-west`, `south`, `south-east`, `static`). *window-gravity* returns a symbol (same as *window-bit-gravity* with `unmap` instead of `forget`). *window-backing-store* returns a symbol (`not-useful`, `when-mapped`, `always`). *window-backing-planes* and *window-backing-pixel* return a pixel. *window-save-under*, *window-map-installed* and *window-override-redirect* return #t or #f. *window-colormap* returns a colormap. *window-map-state* returns a symbol (`unmapped`, `unviewable`, `viewable`). *window-all-event-masks*, *window-your-event-mask*, and *window-do-not-propagate-mask* return a list of symbols (event mask names such as `enter-window`, `pointer-motion`, etc.). All other functions return an integer.

**(drawable-root** *drawable***)** **procedure**
**(drawable-x** *drawable***)** **procedure**
**(drawable-y** *drawable***)** **procedure**
**(drawable-width** *drawable***)** **procedure**
**(drawable-height** *drawable***)** **procedure**
**(drawable-border-width** *drawable***)** **procedure**
**(drawable-depth** *drawable***)** **procedure**

See *XGetGeometry*. Returns the root window, x and y coordinates, width, height, border width, and depth of the specified drawable. *drawable-root* returns a window, all other functions return an integer.

**(map-window** *window***)** **procedure**

See *XMapWindow*.

**(unmap-window** *window***)** **procedure**

See *XUnmapWindow*.

**(destroy-window** *window***)** **procedure**

See *XDestroyWindow*.

**(destroy-subwindows** *window***)** **procedure**

See *XDestroySubwindows*.

**(map-subwindows** *window***)** **procedure**

See *XMapSubwindows*.

**(unmap-subwindows** *window***)** **procedure**

See *XUnmapSubwindows*.

**(circulate-subwindows** *window direction***)** **procedure**

See *XCirculateSubwindows*. *direction* is a symbol (`raise-lowest` or `lower-highest`).

**(clear-window** *window***)** **procedure**

Performs a *clear-area* on the entire window.

**(raise-window** *window***)** **procedure**

See *XRaiseWindow*.

**(lower-window** *window***)** **procedure**

See *XLowerWindow*.

**(restack-windows** *list-of-windows***)** **procedure**

See *XRestackWindows*.

**(query-tree** *window***)** **procedure**

See *XQueryTree*. Returns a list of three elements: root window, parent window, and children (a vector of windows).

**(translate-coordinates** *src-window x y dst-window***)** **procedure**

See *XTranslateCoordinates*. Returns a list of three elements: destination x and y, and child window.

**(query-pointer** *window***)** **procedure**

See *XQueryPointer*. Returns a list of eight elements: x and y, a boolean indicating whether the pointer is on the same screen as the specified window, the root window, the root window's x and y coordinates, the child window, and a list of modifier names (see *grab-button* below).

## 4. Window Property and Selection Functions

**(atom?** *x***)** **procedure**

Returns #t iff *x* is an object of type *atom*.

**(make-atom** *value***)** **procedure**

Returns an atom with the given *value*. *value* is an integer.

**(intern-atom** *display name***)** **procedure**

See *XInternAtom*. *name* is a string or a symbol. The atom is created if it does not yet exist.

**(find-atom** *display name***)** **procedure**

See *XInternAtom*. *name* is a string or a symbol. If the atom does not exist, the symbol `none` is returned.

**(atom-name** *display atom***)** **procedure**

See *XGetAtomName*. Returns a string.

**(list-properties** *window***)** **procedure**

See *XListProperties*. Returns a vector of atoms.

**(get-property** *window property request-type offset length delete?***)** **procedure**

See *XGetWindowProperty*. *property* is an object of type *atom*. *request-type* is an atom or #f in which case *AnyPropertyType* will be used. *offset* and *length* are integers. An error is signaled if *XGetWindowProperty* fails.

*get-property* returns a list of four items: the "actual type" (an atom), the format (an integer), the data (if any, the empty list otherwise), and the number of bytes left (an integer).

The data returned is either a string (if the format indicates 8-bit data) or a vector of integers.

**(change-property** *window property type format mode data***)** **procedure**

See *XChangeProperty*. *property* and *type* are atoms. *format* is an integer (8, 16, or 32). If *format* is 8 *data* must be a string, otherwise a vector of integers of the appropriate size. An error is signaled if the value of *format* is invalid or if *data* holds an integer that exceeds the size indicated by *format*. *mode* is a symbol (`replace`, `prepend`, or `append`).

**(delete-property** *window property***)** **procedure**

See *XDeleteProperty*.

**(rotate-properties** *window vector-of-atoms delta***)** **procedure**

See *XRotateWindowProperties*. *delta* is the amount to rotate (an integer).

**(set-selection-owner!** *display selection owner time***)** **procedure**

See *XSetSelectionOwner*. *selection* is an atom; *owner* is a window; *time* is an integer or the symbol `now` (for *CurrentTime*).

**(selection-owner** *display selection***)** **procedure**

See *XGetSelectionOwner*.

**(convert-selection** *selection target property requestor-window time***)** **procedure**

See *XConvertSelection*. *selection* and *target* are atoms; *property* is an atom or the symbol `none`.

## 5. Colormap Functions

**(color?** *x***)** **procedure**

Returns #t iff *x* is an object of type *color*.

**(make-color** *r g b***)** **procedure**

Returns an object of type *color* with the specified RGB components. *r*, *g*, and *b* are reals in the range 0.0 to 1.0.

**(color-rgb-values** *color***)** **procedure**

Returns a list of three elements, the RGB components of the given color (see *make-color* above).

**(query-color** *colormap pixel***)** **procedure**

See *XQueryColor*.

**(query-colors** *colormap pixels***)** **procedure**

See *XQueryColors*. *pixels* is a vector of pixels. Returns a vector of colors of the same size as *pixels*.

**(lookup-color** *colormap color-name***)** **procedure**

See *XLookupColor*. *color-name* is a string or a symbol. Returns a pair of colors.

**(alloc-color** *colormap color***)** **procedure**

See *XAllocColor*. Returns a pixel (or #f in case of an error).

**(alloc-named-color** *colormap color-name***)** **procedure**

See *AllocNamedColor*. *color-name* is a string or a symbol. Returns a list of three elements: a pixel, and two colors (the closest color and the exact color); or #f in case of an error.

**(colormap?** *x***)** **procedure**

Returns #t iff *x* is an object of type *colormap*.

**(colormap-display** *colormap***)** **procedure**

Returns the display associated with the given colormap.

**(free-colormap** *colormap***)** **procedure**

See *XFreeColormap*.

## 6. Pixel Functions

**(pixel?** *x***)** **procedure**

Returns #t iff *x* is an object of type *pixel*.

**(pixel-value** *pixel***)** **procedure**

Returns the value of the pixel as an unsigned integer.

**(black-pixel** *display***)** **procedure**
**(white-pixel** *display***)** **procedure**

See *XBlackPixel*, *XWhitePixel*. Returns the black/white pixel of the display's default screen.

## 7. Pixmap Functions

**(pixmap?** *x***)** **procedure**

Returns #t iff *x* is an object of type *pixmap*.

**(pixmap-display** *pixmap***)** **procedure**

Returns the display associated with the pixmap.

**(free-pixmap** *pixmap***)** **procedure**

See *XFreePixmap*.

**(create-pixmap** *drawable width height depth***)** **procedure**

See *XCreatePixmap*.

**(create-bitmap-from-data** *window data width height***)** **procedure**

See *XCreateBitmapFromData*. *data* is a string. `(* width height)` must not exceed the number of bits in *string*.

**(create-pixmap-from-bitmap-data** *win data width height foregrnd backgrnd depth***)** **procedure**

See *XCreatePixmapFromBitmapData*. *data* is a string. `(* width height)` must not exceed the number of bits in *string*.

**(read-bitmap-file** *drawable filename***)** **procedure**

See *XReadBitmapFile*. *filename* is a string or a symbol. If *XReadBitmapFile* signals an error, *read-bitmap-file* returns a symbol (`open-failed`, `file-invalid`, or `no-memory`). If it succeeds, *read-bitmap-file* returns a list of five elements: the bitmap (an object of type *pixmap*), the width and height of the bitmap, and the x and y coordinates of the hotspot.

**(write-bitmap-file** *filename pixmap width height x-hot y-hot***)** **procedure**

See *XWriteBitmapFile*. *filename* is a string or a symbol. *x-hot* and *y-hot* are optional (−1 is used if they are omitted), but either both or none of them must be given. *write-bitmap-file* returns a symbol (`success`, `open-failed`, `file-invalid`, or `no-memory`).

## 8. Graphics Context Functions

**(gcontext?** *x***)** **procedure**

Returns #t iff *x* is an object of type *gcontext*.

**(gcontext-display** *gcontext***)** **procedure**

Returns the display associated with the given GC.

**(create-gcontext** . *args***)** **procedure**

See *XCreateGC*. This function is used to create a new GC.

The number of arguments must be even. The 1st, 3rd, etc. argument is the name (a symbol) of an attribute to be set when the graphics context is created, the 2nd, 4th, etc. argument is the corresponding value. The attributes can be specified in any order.

Attributes are *window* (a drawable; mandatory) and all the attributes that can be set by the `set-gcontext-`*attribute*`!` functions below.

**(copy-gcontext** *gcontext drawable***)** **procedure**

See *XCopyGC*. Returns a copy of *gcontext* (associated with the specified drawable).

**(free-gcontext** *gcontext***)** **procedure**

See *XFreeGC*.

**(query-best-size** *display width height shape***)** **procedure**

See *XQueryBestSize*. *shape* is a symbol (`cursor`, `tile`, or `stipple`). Returns a pair of integers (result width and result height).

**(query-best-cursor** *display width height***)** **procedure**
**(query-best-tile** *display width height***)** **procedure**
**(query-best-stipple** *display width height***)** **procedure**

See *XQueryBestSize*. Invokes *query-best-size* with the given arguments and a shape of `cursor`, `tile`, or `stipple`, respectively.

**(gcontext-function** *gcontext***)** **procedure**
**(gcontext-plane-mask** *gcontext***)** **procedure**
**(gcontext-foreground** *gcontext***)** **procedure**
**(gcontext-background** *gcontext***)** **procedure**
**(gcontext-line-width** *gcontext***)** **procedure**
**(gcontext-line-style** *gcontext***)** **procedure**
**(gcontext-cap-style** *gcontext***)** **procedure**
**(gcontext-join-style** *gcontext***)** **procedure**
**(gcontext-fill-style** *gcontext***)** **procedure**
**(gcontext-fill-rule** *gcontext***)** **procedure**
**(gcontext-arc-mode** *gcontext***)** **procedure**
**(gcontext-tile** *gcontext***)** **procedure**
**(gcontext-stipple** *gcontext***)** **procedure**
**(gcontext-ts-x** *gcontext***)** **procedure**
**(gcontext-ts-y** *gcontext***)** **procedure**
**(gcontext-subwindow-mode** *gcontext***)** **procedure**
**(gcontext-exposures** *gcontext***)** **procedure**
**(gcontext-clip-x** *gcontext***)** **procedure**
**(gcontext-clip-y** *gcontext***)** **procedure**
**(gcontext-dash-offset** *gcontext***)** **procedure**

See *XGetGCValues*. Returns the logical operation, plane mask, foreground and background pixel value, line width and style, cap and join style, fill style and rule, arc mode, tiling and stippling pixmap, tiling x- and y-origin, subwindow mode, clipping x- and y-origin, and dashed line information of the specified graphics context.

*gcontext-function* returns a symbol (`clear`, `and`, `and-reverse`, `copy`, `and-inverted`, `no-op`, `xor`, `or`, `nor`, `equiv`, `invert`, `or-reverse`, `copy-inverted`, `nand`, or `set`). *gcontext-plane-mask*, *gcontext-foreground*, and *gcontext-background* return a pixel. *gcontext-tile* and *gcontext-stipple* return a pixmap. The line style is a symbol (`solid`, `dash`, `double-dash`); the cap style is a symbol (`not-last`, `butt`, `round`, `projecting`); the join style is a symbol (`miter`, `round`, `bevel`); the fill style is a symbol (`solid`, `tiled`, `stippled`, `opaque-stippled`); the fill rule is a symbol (`even-odd`, `winding`); the arc mode is a symbol (`chord`, `pie-slice`); the subwindow-mode is a symbol (`clip-by-children`, `include-inferiors`). *gcontext-exposures* returns a boolean. All other functions return an integer.

**(set-gcontext-function!** *gcontext value***)** **procedure**
**(set-gcontext-plane-mask!** *gcontext value***)** **procedure**
**(set-gcontext-foreground!** *gcontext value***)** **procedure**
**(set-gcontext-background!** *gcontext value***)** **procedure**
**(set-gcontext-line-width!** *gcontext value***)** **procedure**

**(set-gcontext-line-style!** *gcontext value***)** **procedure**
**(set-gcontext-cap-style!** *gcontext value***)** **procedure**
**(set-gcontext-join-style!** *gcontext value***)** **procedure**
**(set-gcontext-fill-style!** *gcontext value***)** **procedure**
**(set-gcontext-fill-rule!** *gcontext value***)** **procedure**
**(set-gcontext-arc-mode!** *gcontext value***)** **procedure**
**(set-gcontext-tile!** *gcontext value***)** **procedure**
**(set-gcontext-stipple!** *gcontext value***)** **procedure**
**(set-gcontext-ts-x!** *gcontext value***)** **procedure**
**(set-gcontext-ts-y!** *gcontext value***)** **procedure**
**(set-gcontext-font!** *gcontext value***)** **procedure**
**(set-gcontext-subwindow-mode!** *gcontext value***)** **procedure**
**(set-gcontext-exposures!** *gcontext value***)** **procedure**
**(set-gcontext-clip-x!** *gcontext value***)** **procedure**
**(set-gcontext-clip-y!** *gcontext value***)** **procedure**
**(set-gcontext-clip-mask!** *gcontext value***)** **procedure**
**(set-gcontext-dash-offset!** *gcontext value***)** **procedure**

See *XChangeGC*. Sets the logical operation, plane mask, foreground and background pixel value, line width and style, cap and join style, fill style and rule, arc mode, tiling and stippling pixmap, tiling x- and y-origin, font, subwindow mode, clipping x- and y-origin, clipping bitmap, and dashed line information for the specified graphics context.

The *value* argument to *set-gcontext-font!* is a font, and the *value* argument to *set-gcontext-clip-mask!* is a pixmap. For the types of the *value* argument of the other functions see the return values of the *gcontext-* functions above.

**(set-gcontext-clip-rectangles!** *gcontext x y rectangles ordering***)** **procedure**

See *XSetClipRectangles*. *x* and *y* are integers (the coordinates of the clip-mask origin). *rectangles* is a vector of lists of four integers (x, y, width, and height of each rectangle). *ordering* is a symbol (`unsorted`, `y-sorted`, `yx-sorted`, or `yx-banded`).

**(set-gcontext-dashlist!** *gcontext dash-offset dash-list***)** **procedure**

See *XSetDashes*. *dash-offset* is an integer. *dash-list* is a vector of integers between 0 and 255.

## 9. Graphics Functions

**(clear-area** *window x y width height exposures?***)** **procedure**

See *XClearArea*.

**(copy-area** *src-drawable gcontext src-x src-y width height dst-drawable dst-x dst-y***)** **procedure**

See *XCopyArea*.

**(copy-plane** *src-drawable gcontext plane src-x src-y width height dst-drawable dst-x dst-y***)** **procedure**

See *XCopyPlane*. *plane* is an integer. An error is signaled unless exactly one bit is set in *plane*.

**(draw-point** *drawable gcontext x y***)** **procedure**

See *XDrawPoint*.

**(draw-points** *drawable gcontext vector-of-points relative?***)** **procedure**

See *XDrawPoints*. *vector-of-points* is a vector of pairs consisting of two integers (the x and y coordinates). If *relative?* is #t, *CoordModePrevious* is passed to *XDrawPoints*, otherwise *CoordModeOrigin* is used.

**(draw-line** *drawable gcontext x1 y1 x2 y2***)** **procedure**

See *XDrawLine*.

**(draw-lines** *drawable gcontext vector-of-points relative?***)** **procedure**

See *XDrawLines*. See *draw-points* above.

**(draw-segments** *drawable gcontext vector-of-points***)** **procedure**

See *XDrawSegments*. *vector-of-points* is a vector of lists of four integers (x1, y1, x2, and y2).

**(draw-rectangle** *drawable gcontext x y width height***)** **procedure**

See *XDrawRectangle*.

**(fill-rectangle** *drawable gcontext x y width height***)** **procedure**

See *XFillRectangle*.

**(draw-rectangles** *drawable gcontext vector-of-rectangles***)** **procedure**

See *XDrawRectangles*. *vector-of-rectangles* is a vector of lists of four integers (x, y, width, and height of each rectangle).

**(fill-rectangles** *drawable gcontext vector-of-rectangles***)** **procedure**

See *XFillRectangles*. See *draw-rectangles* above.

**(draw-arc** *drawable gcontext x y width height angle1 angle2***)** **procedure**

See *XDrawArc*.

**(fill-arc** *drawable gcontext x y width height angle1 angle2***)** **procedure**

See *XFillArc*.

**(draw-arcs** *drawable gcontext vector-of-data***)** **procedure**

See *XDrawArcs*. *vector-of-data* is a vector of lists of six integers (x, y, width, height, angle1, and angle2).

**(fill-arcs** *drawable gcontext vector-of-data***)** **procedure**

See *XFillArcs*. See *draw-arcs* above.

**(fill-polygon** *drawable gcontext vector-of-points relative? shape***)** **procedure**

See *XFillPolygon*. See *draw-points* above. *shape* is a symbol (`complex`, `non-convex`, or `convex`).

## 10. Font Functions

**(font?** *x***)** **procedure**

Returns #t iff *x* is an object of type *font*.

**(font-display )** **procedure**

Returns the display associated with the given font.

**(open-font** *display font-name***)** **procedure**

See *XLoadQueryFont*. *font-name* is a string or a symbol.

**(close-font** *font***)** **procedure**

See *XUnloadFont*.

**(font-name** *font***)** **procedure**

Returns the name of the specified font (a string) or #f if the name could not be determined (e. g. when the font has been obtained by a call to *gcontext-font*).

**(gcontext-font** *gcontext***)** **procedure**

Calls *XQueryFont* with the GC obtained by *XGContextFromGC*. Only a limited number of functions can be applied to a font returned by *gcontext-font*, since it has neither a name nor a font-ID.

**(list-font-names** *display pattern***)** **procedure**

See *XListFonts*. *pattern* is a string or a symbol. Returns a vector of font names (strings).

**(list-fonts** *display pattern***)** **procedure**

See *XListFontsWithInfo*. *pattern* is a string or a symbol. Returns a vector of fonts. These fonts are ‘‘pseudo fonts’’ which do not have a font-ID. A pseudo font is loaded automatically and turned into a ‘‘real’’ font the first time it is passed to a function that makes use of the font-ID.

**(font-direction** *font***)** **procedure**
**(font-min-byte2** *font***)** **procedure**
**(font-max-byte2** *font***)** **procedure**
**(font-min-byte1** *font***)** **procedure**
**(font-max-byte1** *font***)** **procedure**
**(font-all-chars-exist?** *font***)** **procedure**
**(font-default-char** *font***)** **procedure**
**(font-ascent** *font***)** **procedure**
**(font-descent** *font***)** **procedure**

These functions return the font direction as a symbol (`left-to-right` or `right-to-left`), the first and last character (as an integer), the first and last row (integer), an indication whether all characters have non-zero size (boolean), the default character (integer), and the ascent and descent (integer) of the specified font.

**(char-rbearing** *font index***)** **procedure**
**(char-lbearing** *font index***)** **procedure**
**(char-width** *font index***)** **procedure**
**(char-ascent** *font index***)** **procedure**
**(char-descent** *font index***)** **procedure**

These functions return the metrics of the character specified by the integer *index* of the given font. Each function returns an integer. *font* can be a 1-byte as well as a 2-byte font.

**(max-char-lbearing** *font***)** **procedure**
**(max-char-rbearing** *font***)** **procedure**
**(max-char-width** *font***)** **procedure**
**(max-char-ascent** *font***)** **procedure**
**(max-char-descent** *font***)** **procedure**

These functions return the maximum metrics over all characters in the specified font. Each function returns an integer.

**(min-char-lbearing** *font***)** **procedure**
**(min-char-rbearing** *font***)** **procedure**
**(min-char-width** *font***)** **procedure**
**(min-char-ascent** *font***)** **procedure**
**(min-char-descent** *font***)** **procedure**

These functions return the minimum metrics over all characters in the specified font. Each function returns an integer.

**(font-properties** *font***)** **procedure**

Returns a vector of font properties; each element of the vector is a pair consisting of the property name (an atom) and an unsigned integer (the value of the property).

**(font-property** *font property-name***)** **procedure**

Returns the value of the specified property associated with the specified font. *property-name* is a string or a symbol.

**(font-path** *display***)** **procedure**

See *XGetFontPath*. Returns the current font path as a vector of strings.

**(set-font-path!** *display path***)** **procedure**

See *XSetFontPath*. *path* is a list; each element is a string or a symbol.

## 11. Text Metrics and Text Drawing Functions

**(text-width** *font text format***)** **procedure**

See *XTextWidth*, *XTextWidth16*. *format* indicates whether 8-bit or 16-bit text is used; it is either the symbol `1-byte` or the symbol `2-byte`. *text* is a vector of integers; the integers must not exceed the size indicated by the format.

**(extents-lbearing** *font text format***)** **procedure**
**(extents-rbearing** *font text format***)** **procedure**
**(extents-width** *font text format***)** **procedure**
**(extents-ascent** *font text format***)** **procedure**
**(extents-descent** *font text format***)** **procedure**

See *XTextExtents*, *XTextExtents16*. These functions are used to compute the overall metrics of an 8-bit or 16-bit character string. Each function returns an integer. For the format of *text* and *format* see *text-width* above.

**(draw-image-text** *drawable gcontext x y text format***)** **procedure**

See *XDrawImageString*, *XDrawImageString16*. See *text-width* above.

**(draw-poly-text** *drawable gcontext x y text format***)** **procedure**

See *XDrawText*, *XDrawText16*. See *text-width* above. *text* is a vector of integers with intermixed objects of type *font*.

**(translate-text** *string***)** **procedure**

Converts the string into a representation suitable as an argument to *text-width*, *draw-image-text*, or *draw-poly-text* (a vector of integers obtained by applying *char->integer* to the characters of the string argument).

## 12. Cursor Functions

**(cursor?** *x***)** **procedure**

Returns #t iff *x* is an object of type *cursor*.

**(cursor-display** *cursor***)** **procedure**

Returns the display associated with the given cursor.

**(free-cursor )** **procedure**

See *XFreeCursor*.

**(create-cursor** *src mask x y foreground background***)** **procedure**

See *XCreatePixmapCursor*. *src* and *mask* are pixmaps. *mask* can be the symbol `none`.

**(create-glyph-cursor** *src src-char mask mask-char foreground background***)** **procedure**

See *XCreateGlyphCursor*. *src* and *mask* are fonts. *mask* can be the symbol `none`. The display is obtained from *src*. *src-char* and *mask-char* are integers.

**(create-font-cursor** *display src-char***)** **procedure**

See *XCreateGlyphCursor*. Calls *create-glyph-cursor* with the font named ‘‘cursor’’, the specified *src-char*, a *mask-char* of `(1+ src-char)`, black foreground, and white background.

**(recolor-cursor** *cursor foreground background***)** **procedure**

See *XRecolorCursor*

**(define-cursor** *window cursor***)** **procedure**

Synonym for `(set-window-cursor! window cursor)`.

**(undefine-cursor** *window***)** **procedure**

Synonym for `(set-window-cursor! window 'none)`.

## 13. Grab Functions

**(grab-pointer** *window owner? events ptr-sync? kbd-sync? confine-to cursor time***)** **procedure**

See *XGrabPointer*. *window* and *confine-to* are windows. *events* is a list of symbols (event mask names, such as `enter-window`, `pointer-motion`, etc.). *ptr-sync?* and *kbd-sync?* determine whether synchronous or asynchronous grab mode is to be used. *time* is an integer or the symbol `now` (for *CurrentTime*). *grab-pointer* returns a symbol (`success`, `not-viewable`, `already-grabbed`, `frozen`, or `invalid-time`).

**(ungrab-pointer** *display time***)** **procedure**

See *XUngrabPointer*.

**(grab-button** *win button mod owner? events ptr-sync? kbd-sync? confine-to cursor***)** **procedure**

See *XGrabButton*. *button* is a symbol (`button1` ... `button5`, or `any-button`). `mod` (modifiers) is a list of symbols (`shift`, `lock`, `control`, `mod1` ... `mod5`, `button1` ... `button5`, or `any-modifier`). For the other arguments see *grab-pointer* above.

**(ungrab-button** *window button modifiers***)** **procedure**

See *XUngrabButton*. See *grab-button* above.

**(change-active-pointer-grab** *display events cursor time***)** **procedure**

See *XChangeActivePointerGrab*. *events* is a list of symbols (event mask names, such as `enter-window`, `pointer-motion`, etc.).

**(grab-keyboard** *window owner? pointer-sync? keyboard-sync? time***)** **procedure**

See *XGrabKeyboard*. For a description of the arguments and the return value see *grab-pointer* above.

**(ungrab-keyboard** *display time***)** **procedure**

See *XUngrabKeyboard*.

**(grab-key** *window key modifiers owner? pointer-sync? keyboard-sync?***)** **procedure**

See *XGrabKey*. *key* is a keycode (an integer) or the symbol `any`. For the other arguments see *grab-pointer* above.

**(ungrab-key** *window key modifiers***)** **procedure**

See *XUngrabKey*. See *grab-key* above.

**(allow-events** *display mode time***)** **procedure**

See *XAllowEvents*. *mode* is a symbol (`async-pointer`, `sync-pointer`, `replay-pointer`, `async-keyboard`, `sync-keyboard`, `replay-keyboard`, `async-both`, or `sync-both`).

**(grab-server** *display***)** **procedure**

See *XGrabServer*.

**(ungrab-server** *display***)** **procedure**

See *XUngrabServer*.

**(with-server-grabbed** *display . body-forms***)** **syntax**

This macro performs a *grab-server* on the specified display, evaluates the *body-forms* in order, and then ungrabs the server. The macro body is guarded by a *dynamic-wind* to ensure that the *ungrab-server* is performed when a body-form calls a continuation created outside the macro, and that it is grabbed again when the body is re-entered at a later point in time. *with-server-grabbed* returns the value of the last body-form.

## 14. Window Manager Functions

**(reparent-window** *window parent-window x y***)** **procedure**

See *XReparentWindow*.

**(install-colormap** *colormap***)** **procedure**

See *XInstallColormap*.

**(uninstall-colormap** *colormap***)** **procedure**

See *XUninstallColormap*.

**(list-installed-colormaps** *window***)** **procedure**

See *XListInstalledColormaps*. Returns a vector of colormaps.

**(set-input-focus** *display window revert-to time***)** **procedure**

See *XSetInputFocus*. *window* can be the symbol `pointer-root`. *revert-to* is a symbol (`none`, `pointer-root`, or `parent`). *time* is an integer or the symbol `now`.

**(input-focus** *display***)** **procedure**

See *XGetInputFocus*. Returns a pair the car of which is a window, and the cdr is a symbol (`none`, `pointer-root`, or `parent`).

**(general-warp-pointer** *display dst-win dst-x dst-y src-win src-x src-y src-width src-height***)** **procedure**

See *XWarpPointer*.

**(warp-pointer** *dst-window dst-x dst-y***)** **procedure**

See *XWarpPointer*. Invokes *general-warp-pointer* with the display associated with the *dst-window*, the *dst-window*, *dst-x*, *dst-y*, a *src-window* of `none`, and zero source coordinates and dimensions.

**(warp-pointer-relative** *display x-offset y-offset***)** **procedure**

See *XWarpPointer*. Invokes *general-warp-pointer* with the specified *display*, a *dst-window* of `none`, *x-offset*, *y-offset*, a *src-window* of `none`, and zero source coordinates and dimensions.

**(bell** *display . percent***)** **procedure**

See *XBell*. *percent* is an integer between -100 and 100. If *percent* is omitted, 0 is used.

**(set-access-control** *display enable?***)** **procedure**

See *XSetAccessControl*.

**(change-save-set** *window mode***)** **procedure**

See *XChangeSaveSet*. *mode* is a symbol (`insert` or `delete`).

**(set-close-down-mode** *display mode***)** **procedure**

See *XSetCloseDownMode*. *mode* is a symbol (`destroy-all`, `retain-permanent`, or `retain-temporary`).

**(get-pointer-mapping** *display***)** **procedure**

See *XGetPointerMapping*. Returns a vector of 256 integers.

**(set-pointer-mapping** *display mapping***)** **procedure**

See *XSetPointerMapping*. *mapping* is a vector of integers. Returns #t if *XSetPointerMapping* succeeds, #f otherwise.

## 15. Event Handling Functions

**(event-listen** *display wait?***)** **procedure**

See *XPending*, *XPeekEvent*. Returns the size of the display's event queue. If *wait?* is true and the event queue is empty, *event-listen* flushes the output buffer and blocks until an event is received from the server.

**(get-motion-events** *window from-time to-time***)** **procedure**

See *XGetMotionEvents*. *from-time* and *to-time* are integers or the symbol `now`. *get-motion-events* returns a vector of lists of three elements: a time stamp (an integer or the symbol `now`), and the x and y coordinates (integers).

**(handle-events** *display discard? peek? . clauses***)** **syntax**

See *XNextEvent*, *XPeekEvent*, *XIfEvent*, *XPeekIfEvent*. *handle-events* is a special form. Each *clause* is of the form *(guard function)*; *guard* is either an event name (a symbol, e. g. `key-press` or `exposure`), a list of event names, or the symbol `else`. *handle-events* gets the next event from the specified display. Then the event type is matched against each event name in each guard in order. When a match occurs, the corresponding function is invoked with the name of the event being dispatched (a symbol) and other, event specific arguments (see below). When no clause matches and an `else` clause is present, the function from this clause is invoked. *handle-events* loops until a function returns a value not equal to #f in which case handle-events returns this value.

If *discard?* is true, unprocessed events (i. e. events for which no matching clause has been found) are removed from the event queue, otherwise they are left in place. If *peek?* is true, processed events are not removed from the event queue.

The following list gives all event specific arguments for each event type. The first argument is always the event type (a symbol).

In the following list, arguments with names of the form *something-window* (or simply *window*) are always of type *window*; arguments with names of the form *something-atom* (or simply *atom*) are always of type *atom*. *time* is an integer or the symbol `now`. *x*, *y*, *width*, *height*, *border-width*, *x-root*, *y-root*, *count*, *major-code*, *minor-code*, and *keycode* are integers. *state* is a list of symbols (`shift`, `lock`, `control`, `mod1` ... `mod5`, `button1` ... `button5`). *button* is one of the symbols `button1` ... `button5`, *button-mask* is a list of one or more of these symbols. *cross-mode* is a symbol (`normal`, `grab`, `ungrab`). *place* is a symbol (`top` or `bottom`).

**key-press, key-release:**
*window*, *root-window*, *sub-window*, *time*, *x*, *y*, *x-root*, *y-root*, *state*, *keycode*, *same-screen?*.

**button-press, button-release:**
*window*, *root-window*, *sub-window*, *time*, *x*, *y*, *x-root*, *y-root*, *state*, *button*, *same-screen?*.

**motion-notify:**
*window*, *root-window*, *sub-window*, *time*, *x*, *y*, *x-root*, *y-root*, *state*, *is-hint?*, *same-screen?*.

**enter-notify, leave-notify:**
*window*, *root-window*, *sub-window*, *time*, *x*, *y*, *x-root*, *y-root*, *cross-mode*, *cross-detail* (one of the symbols `ancestor`, `virtual`, `inferior`, `nonlinear`, `nonlinear-virtual`), *same-screen?*, *focus?*, *button-mask*.

**focus-in, focus-out:**
*window*, *cross-mode*, *focus-detail* (one of the symbols `ancestor`, `virtual`, `inferior`, `nonlinear`, `nonlinear-virtual`, `pointer`, `pointer-root`, `none`).

**keymap-notify:**
*window*, *keymap* (a string of length 32).

**expose:**
*window*, *x*, *y*, *width*, *height*, *count*.

**graphics-expose:**
*window*, *x*, *y*, *width*, *height*, *count*, *major-code*, *minor-code*.

**no-expose:**
*window*, *major-code*, *minor-code*.

**visibility-notify:**
*window*, *visibility-state* (one of the symbols `unobscured`, `partially-obscured`, `fully-obscured`).

**create-notify:**
*parent-window*, *window*, *x*, *y*, *width*, *height*, *border-width*, *override-redirect?*.

**destroy-notify:**
*event-window*, *window*.

**unmap-notify:**
*event-window*, *window*, *from-configure*.

**map-notify:**
*event-window*, *window*, *override-redirect*.

**map-request:**
*parent-window*, *window*.

**reparent-notify:**
*event-window*, *parent-window*, *window*, *x*, *y*, *override-redirect*.

**configure-notify:**
*event-window*, *window*, *x*, *y*, *width*, *height*, *border-width*, *above-window*, *override-redirect?*.

**configure-request:**
*parent-window*, *window*, *x*, *y*, *width*, *height*, *border-width*, *above-window*, *stack-mode* (see *set-window-stack-mode!* above), *value-mask* (an integer).

**gravity-notify:**
*event-window*, *window*, *x*, *y*.

**resize-request:**
*window*, *width*, *height*.

**circulate-notify:**
*event-window*, *window*, *place*.

**circulate-request:**
*parent-window*, *window*, *place*.

**property-notify:**
*window*, *atom*, *time*, *property-state* (one of the symbols `new-value`, `deleted`).

**selection-clear:**
*window*, *selection-atom*, *time*.

**selection-request:**
*owner-window*, *requestor-window*, *selection-atom*, *target-atom*, *property-atom*, *time*.

**selection-notify:**
*requestor-window*, *selection-atom*, *target-atom*, *property-atom*, *time*.

**colormap-notify:**
*window*, *colormap*, *new?*, *colormap-installed?*.

**client-message:**
*window*, *message type* (an atom), *message data* (a string of length 20, or a vector of 10 or 5 integer numbers, or, if the format field of the event is wrong, the format as a number).

**mapping-notify:**
*window*, *request* (one of the symbols `modifier`, `keyboard`, `pointer`), *keycode*, *count*.

## 16. Inter-Client Communication Functions

**(iconify-window** *window screen-number***)** **procedure**

See *XIconifyWindow*.

**(withdraw-window** *window screen-number***)** **procedure**

See *XWithdrawWindow*.

**(reconfigure-wm-window** . *args***)** **procedure**

See *XReconfigureWMWindow*.

For the format of the arguments see *create-window* above. Mandatory attributes are *window* and *screen-number* (an integer). Optional attributes are *x*, *y*, *width*, *height border-width* (integers), *sibling* (a window), and *stack-mode* (a symbol; one of `above`, `below`, `top-if`, `bottom-if`, `opposite`).

**(get-text-property** *window atom***)** **procedure**

See *XGetTextProperty*. Returns a text property as a list of strings or #f if the specified property does not exist.

**(set-text-property!** *window value atom***)** **procedure**

See *XSetTextProperty*. *value* is a list holding the items of the text property (strings or symbols).

**(wm-protocols** *window***)** **procedure**

See *XGetWMProtocols*. Returns a vector of atoms.

**(set-wm-protocols!** *window protocols***)** **procedure**

See *XSetWMProtocols*. *protocols* is a vector of atoms.

**(wm-name** *window***)** **procedure**

See *XGetTextProperty*. Returns the WM_NAME property as a list of strings or #f if it does not exist.

**(set-wm-name!** *window name***)** **procedure**

See *XSetTextProperty*. *name* is a list of strings or symbols.

**(wm-icon-name** *window***)** **procedure**

See *XGetTextProperty*. Returns the WM_ICON_NAME property as a list of strings or #f if it does not exist.

**(set-wm-icon-name!** *window name***)** **procedure**

See *XSetTextProperty*. *name* is a list of strings or symbols.

**(wm-client-machine** *window***)** **procedure**

See *XGetTextProperty*, *XGetWMClientMachine*. Returns the WM_CLIENT_MACHINE property as a list of strings or #f if it does not exist.

**(set-wm-client-machine!** *window value***)** **procedure**

See *XSetTextProperty*, *XSetWMClientMachine*. *value* is a list of strings or symbols.

**(wm-class** *window***)** **procedure**

See *XGetClassHint*. Returns a pair (name and class) each component of which is either a string or #f.

**(set-wm-class!** *window name class***)** **procedure**

See *XSetClassHint*. *name* and *class* are strings or symbols.

**(wm-command** *window***)** **procedure**

See *XGetCommand* (in X11 Release 4 or newer releases). Returns the value of the WM_COMMAND property of the given window as a list of strings.

**(set-wm-command!** *window command***)** **procedure**

See *XSetCommand*. *command* is a list; each element is either a string or a symbol.

**(transient-for** *window***)** **procedure**

See *XGetTransientForHint*. Returns a window.

**(set-transient-for!** *window property-window***)** **procedure**

See *XSetTransientForHint*.

**(wm-normal-hints** *window***)** **procedure**

See *XGetWMSizeHints*. Returns a list of hints. Each element is set to the empty list if the corresponding hint has not been set for the specified window.

The elements of the list correspond to the following hints (in this order): *x*, *y*, *width*, and *height* (program specified); *x*, *y*, *width* and *height* (user specified); *min-width* and *min-height*; *max-width* and *max-height*; *width-inc* and *height-inc*; *min-aspect-x*, *min-aspect-y*, *max-aspect-x* and *max-aspect-y*; *base-width* and *base-height*; and *gravity*. All elements are integers except for the value of *gravity* which is a symbol (see the *window-gravity* procedure above).

**(set-wm-normal-hints!** . *args*) **procedure**

See *XSetWMSizeHints*. For the format of the arguments see *create-window* above. Attributes are *window* (mandatory) and the names of the hints listed under *wm-normal-hints* above.

**(wm-hints** *window*) **procedure**

See *XGetWMHints*. Returns a list of hints. Each element is set to the empty list if the corresponding hint has not been set for the specified window.

The elements of the list correspond to the following hints (in this order): *input?*, *initial-state*, *icon-pixmap*, *icon-window*, *icon-x*, *icon-y*, *icon-mask*, and *window-group*. The value of *input?* is a boolean. *initial-state* is a symbol (`dont-care`, `normal`, `zoom`, `iconic`, `inactive`). The values of *icon-pixmap* and *icon-mask* are pixmaps. *icon-window* and *window-group* are windows. *icon-x* and *icon-y* are integers.

**(set-wm-hints!** . *args*) **procedure**

See *XSetWMHints*. For the format of the arguments see *create-window* above. Attributes are *window* (mandatory) and the names of the hints listed under *wm-hints* above.

**(icon-sizes** *window*) **procedure**

See *XGetIconSizes*. Returns a vector of lists of six integers (*min-width*, *min-height*, *max-width*, *max-height*, *width-inc*, and *height-inc*).

**(set-icon-sizes!** *window icon-sizes*) **procedure**

See *XSetIconSizes*. *icon-sizes* is a vector of lists of six integers (see *icon-sizes* above).

## 17. Keyboard Utility Functions

**(display-min-keycode** *display*) **procedure**
**(display-max-keycode** *display*) **procedure**

Returns the minimum/maximum keycode (an integer) for the given display.

**(display-keysyms-per-keycode** *display*) **procedure**
**Returns the number of keysyms per keycode for the given display.**

**(string–>keysym** *string*) **procedure**

See *XStringToKeysym*. *string* is a string or a symbol. Returns an integer if *XStringToKeysym* succeeds, #f otherwise.

**(keysym–>string** *keysym*) **procedure**

See *XKeysymToString*. *keysym* is an integer. Returns #f if *XKeysymToString* fails.

**(keycode−>keysym** *display keycode index***)** **procedure**

See *XKeycodeToKeysym*. *keycode* and *index* are integers.

**(keysym−>keycode** *display keysym***)** **procedure**

See *XKeysymToKeycode*. *keysym* is an integer.

**(lookup-string** *display keycode mask***)** **procedure**

See *XLookupString*. *keycode* is an integer. *mask* is a list of symbols (`shift`, `lock`, `control`, `mod1` ... `mod5`, `button1` ... `button5`, or `any−modifier`).

**(rebind-keysym** *display keysym modifiers string***)** **procedure**

See *XRebindKeysym*. *keysym* is an integer. *modifiers* is a vector of integers.

**(refresh-keyboard-mapping** *window type***)** **procedure**

See *XRefreshKeyboardMapping*. *type* is a symbol (`modifier`, `keyboard`, or `pointer`). Invokes *XRefreshKeyboardMapping* with a faked event structure holding the specified window and request type.

## 18. Other Utility Functions

**(xlib-release-4-or-later? )** **procedure**

Returns always #t.

**(xlib-release-5-or-later? )** **procedure**

Returns #t iff the Xlib extension is linked together with the X11 Release 5 Xlib or later versions of the Xlib.

**(xlib-release-6-or-later? )** **procedure**

Returns #t iff the Xlib extension is linked together with the X11 Release 6 Xlib or later versions of the Xlib.

**(get-default** *display program option***)** **procedure**

See *XGetDefault*. *program* and *option* are strings or symbols. Returns a string of #f if the option does not exist for the specified program.

**(resource-manager-string** *display***)** **procedure**

See *XResourceManagerString*. Returns a string or #f if the RESOURCE_MANAGER property does not exist on the root window.

**(parse-geometry** *string***)** **procedure**

See *XParseGeometry*. Returns a list of six elements: two booleans indicating whether x or or y are negative and four integers (x, y, width, and height). Each of the elements can be #f to indicate that the respective value was not found in the string.

**(parse-color** *colormap string***)** **procedure**

See *XParseColor*. Returns an object of type *color* or #f if *XParseColor* fails.

**(store-buffer** *display bytes buffer***)** **procedure**

See *XStoreBuffer*. *bytes* is a string; *buffer* is an integer between 0 and 7.

**(store-bytes** *display bytes***)** **procedure**

See *XStoreBytes*. *bytes* is a string.

**(fetch-buffer** *display buffer***)** **procedure**

See *XFetchBuffer*. *buffer* is an integer between 0 and 7. Returns a string.

**(fetch-bytes** *display***)** **procedure**

See *XFetchBytes*. Returns a string.

**(rotate-buffers** *display delta***)** **procedure**

See *XRotateBuffers*. *delta* is an integer (the amount to rotate the buffers).

**(with** *object . body-forms***)** **syntax**

*with* is a macro. *object* must be a drawable, a graphics context, or a font. The *body-forms* are evaluated in order; *with* returns the value of the last body-form.

Within the scope of the *with*, the first call to an accessor function accessing *object* (such as `window-`*attribute* or `font-`*attribute*) causes the result of the corresponding Xlib function to be retained in a cache; subsequent calls just return the value from the cache. Likewise, calls to Xlib functions for mutator functions modifying *object* (such as `set-window-`*attribute*`!`) are delayed until exit of the *with* body or until an accessor function is called and the cached data for this accessor function has been invalidated by the call to a mutator function.

## 19. Server Extension Functions

**(list-extensions** *display***)** **procedure**

See *XListExtensions*. Returns a vector of strings.

**(query-extension** *display name***)** **procedure**

See *XQueryExtension*. *name* is a string or a symbol. Returns a list of three elements: the major opcode (an integer) or #f if the extension has no major opcode, the base event type code (an integer) of #f if the extension has no additional event types, and the base error code (an integer) of #f if the extension has no additional error codes. *query-extension* returns #f if the specified extension is not present.

### 20. Error Handling

**x-error-handler** **variable**

See *XSetErrorHandler*. If an error event is received and the global variable *x-error-handler* is bound to a compound procedure, this procedure is invoked with the following arguments: a display, the serial number of the failed request (an integer), the error code (either an integer or one of the symbols `bad-request`, `bad-value`, `bad-window`, `bad-pixmap`, `bad-atom`, `bad-cursor`, `bad-font`, `bad-match`, `bad-drawable`, `bad-access`, `bad-alloc`, `bad-color`, `bad-gcontext`, `bad-id-choice`, `bad-name`, `bad-length`, or `bad-implementation`), the major and minor op-code of the failed request (integers), and a resource-ID (an integer).

If an error event is received and this variable is not bound to a compound procedure, the Xlib default error handler is invoked. The initial value of this variable is the empty list.

**x-fatal-error-handler** **variable**

See *XSetIOErrorHandler*. If a fatal I/O error occurs and the global variable *x-fatal-error-handler* is bound to a compound procedure, this procedure is invoked with a display as argument. The procedure must invoke *exit*. If a fatal error occurs and this variable is not bound to a compound procedure, or if the procedure returns, the Xlib default fatal error handler is invoked and the interpreter terminates with an exit code of 1. The initial value of this variable is the empty list.

### 21. Interaction with the Garbage Collector

The Scheme garbage collector destroys objects of type *colormap*, *cursor*, *display*, *font*, *gcontext*, *pixmap*, or *window* that are not longer accessible from within the Scheme program. This is done by invoking the function *free-colormap*, *free-cursor*, *close-display*, *close-font*, *free-gcontext*, *free-pixmap*, or *destroy-window*, respectively, with the object to be destroyed as an argument.

The garbage collector only destroys objects that have been created from with the Scheme program (by functions like *create-pixmap* or *open-display*). Objects that have been obtained from the Xlib through functions like *display-default-colormap* (and are owned by the Xlib internals), are ignored by the garbage collector.

Programmers must make sure that an object is accessible during the object's entire lifetime, otherwise future runs of the garbage collector can result in undesired termination of the object. One must be especially careful when results of functions that create new objects (such as *create-window*) are ignored or assigned to local variables as in

```
(define dpy (open-display))
(define root (display-root-window dpy))

(do ((x 0 (+ x 10)) (y 0 (+ y 10))) ((= x 50))
  (let ((win
          (create-window 'parent root 'x x 'y y 'width 20 'height 20)))
    (manage-window win)))
```

In this example, after termination of the do-loop, the garbage collector will destroy the newly created windows, as they are not accessible from within the program. If this is not desired, the windows could be put into a variable (for instance, be *consed* into a list) that is defined outside of the body of the loop.

# Index

## E



## F

## X

# Table of Contents