Interface from GCL to X Windows

Gordon S. Novak Jr.
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

1 Introduction

This document describes a relatively easy-to-use interface between XGCL (X version of Gnu Common Lisp) and X windows. The interface consists of several parts:

1. Hiep Huu Nguyen has written (and adapted from X Consortium software) an interface between GCL and Xlib, the X library in C. Xlib functions can be called directly if desired, but most users will find the `dwindow` functions easier to use. There is little documentation of these functions, but the Xlib documentation can be consulted, and the `dwindow` functions can be examined as examples.

2. The `dwindow` functions described in this document, which call the Xlib functions and provide an easier interface for Lisp programs.

3. It is possible to make an interactive graphical interface within a web page; this is described in a section below.

The source file for the interface (written in GLISP) is `dwindow.lsp`; this file is compiled into a file in plain Lisp, `dwtrans.lsp`. `dwtrans.lsp` is compiled as part of XGCL.

The functions in this package use the convention that the coordinate `(0 0)` is the lower-left corner of a window, with positive `y` being upward. This is different from the convention used by X, which assumes that `(0 0)` is the upper left corner and that positive `y` is downward.

In the descriptions below, some function arguments are shown with a type, e.g. `arg:type`, to indicate the expected type of the argument. The type `vector` is a list `(x y)` of integers. The argument `w` that is used with many functions is of type `window` (`window` is a Lisp data structure used by the `dwindow` functions).

Both the Xlib and `dwindow` functions are in the package `xlib:`. In order to use these functions, the Lisp command `(use-package 'xlib)` should be used to import the `dwindow` symbols.

2 Examples and Utilities

2.1 dwtest

The file `dwtest.lsp` contains example functions that illustrate the use of the `dwindow` package. The function call `(wtesta)` creates a small window for testing. `(wtestb)` through `(wtestk)` perform drawing and mouse interaction tests using the window. These functions may be consulted as examples of the use of commonly used `dwindow` functions.

2.2 pcalc

The file `pcalc.lsp` implements a pocket calculator as a `picmenu`; its entry is `(pcalc)`.

2.3 draw

The file `drawtrans.lsp` contains an interactive drawing program; its entry is `(draw 'foo)` where `foo` is the name of the drawing. The file `ice-cream.lsp` can be loaded, followed by `(draw 'ice-cream)` to examine an example drawing. `draw` can produce a Lisp program or a set of LaTeX commands to recreate the drawing; use `origin to zero` before making a program. `(draw-out file names)` will write definitions of drawings in the list `names` to the file `file`.

2.4 editors

The file `editorstrans.lsp` contains some interactive editing programs; it is a translation of the file `editors.lsp` . One useful editor is the color editor; after entering `(wtesta)` (in file `dwtest.lsp`), enter `(edit-color myw)` to edit a color. The result is an `rgb` list as used in `window-set-color`.

A simple line editor and an Emacs-like text editor are described in sections 6.2 and 6.3 below.

3 Menus

The function `menu` provides an easy interface to make a pop-up menu, get a selection from it, and destroy it:

```
(menu items &optional title)
```

Example: `(menu '(red white blue))`

This simple call is all that is needed in most cases. More sophisticated menu features are described below.

The `items` in a menu is a list; each item may be a symbol, a `cons` of a symbol or string and the corresponding value, or a `cons` of a function name and the corresponding value. In the latter case, the function is expected to draw the corresponding menu item.

If a function name is specified as the first element of a menu item, the drawing function should have arguments `(fn w x y)`, where `w` is the window and `x` and `y` are the lower-left corner of the drawing area. The property list of the function name should have the property `display-size`, which should be a list `(width height)` in pixels of the displayed symbol.

Menus can be associated with a particular window; if no window is specified, the menu is associated with the window where the mouse cursor is located when the menu is initialized (which might not be a Lisp user's window). If a menu is associated with a user window, it may be *permanent* (left displayed after a selection is made) and may be *flat* (drawn directly on the containing window, rather than having its own window).

A menu can be created by `menu-create` :

```
(menu-create items &optional title w:window x y perm flat font)
```

`title`, if specified, is displayed over the menu. `w` is an existing `window`; if specified, the menu is put within this window at the `x y` offsets specified (adjusted if necessary to keep the menu inside the window). If no `w` is specified, or if `x` is `nil`, the menu is put where the cursor is the first time the menu is displayed. `perm` is non-`nil` if the menu is to be permanent, *i.e.*, is to be left displayed after a selection has been made. `flat` is non-`nil` if the menu is to be drawn directly on the containing window. `font` is a symbol or string that names the font to be used; the default is a `9x15` typewriter font.

The menu is returned as the value of `menu-create`. Such a menu can be saved; selections can be made from a menu `m` as follows:

```
(menu-select m &optional inside)    or        (menu-select!  m)
```

`menu-select` will return `nil` if the mouse is clicked outside the menu, or is moved outside after it has been inside (or if `inside` is not `nil`), provided that the menu is contained within a user-created window. `menu-select!` requires that a choice be made.

In order to avoid wasting storage, unused menus should be destroyed: `(menu-destroy m)`. The simple `menu` function destroys its menu after it is used.

```
(menu-size m)
(menu-moveto-xy m x y)
(menu-reposition m)
```

`menu-reposition` will reposition a `flat` menu within its parent window by allowing the user to position a ghost box using the mouse. `menu-size` returns the size of the menu as a vector, `(x y)`. `menu-moveto-xy` adjusts the offsets to move a `flat` menu to the specified

position within its parent window. These functions and `menu-destroy` work for picmenus and barmenus as well.

```
(menu-item-position m name &optional location)
```

`menu-item-position` returns a vector `(x y)` that gives the coordinates of the menu item whose name is `name`. `location` may be `center`, `left`, `right`, `top`, or `bottom`; the default is the lower-left corner of the menu item. `center` specifies the center of the box containing the menu item; the other `location` values are at the center of the specified edge of the box.

3.1 Picmenus

A `picmenu` (picture menu) is analogous to a menu, but involves a user-defined picture containing sensitive spots or "buttons". The test function `(wteste)` shows an example of a `picmenu`. A `picmenu` is created by:

```
(picmenu-create buttons width height drawfn
            &optional title dotflg w:window x y perm flat font boxflg)
```

If a picmenu is to be used more than once, the common parts can be made into a `picmenu-spec` and reused:

```
(picmenu-create-spec buttons width height drawfn
            &optional dotflg font)

(picmenu-create-from-spec spec:picmenu-spec
            &optional title w:window x y perm flat boxflg)
```

`width` and `height` are the size of the area occupied by the picture. `(drawfn w x y)` should draw the picture at the offset `x y`. Note that the `draw` utility can be used to make the drawing function, including `picmenu` buttons. `dotflg` is non-`nil` if it is desired that small boxes be automatically added to the sensitive points when the picture is drawn. `boxflg` is non-`nil` if a box is to be drawn around the picmenu when the picture is drawn (this is only needed for flat picmenus). If `perm` is non-nil, the drawing program is not called when a selection is to be made, so that an external program must draw the `picmenu`; this avoids the need to redraw a complex picture. The remaining arguments are as described for menus.

Each of the `buttons` in a picmenu is a list:

```
(buttonname offset size highlightfn unhighlightfn)
```

`buttonname` is the name of the button; it is the value returned when that button is selected. `offset` is a vector `(x y)` that gives the offset of the center of the button from the lower-left corner of the picture. The remainder of the button list may be omitted. `size` is an optional list `(width height)` that gives the size of the sensitive area of the button; the default size is `(12 12)`. `(highlightfn w x y)` and `(unhighlightfn w x y)` (where `(x y)` is the center

of the button in the coordinates of `w`) are optional functions to highlight the button area when the cursor is moved into it and unhighlight the button when the cursor is moved out; the default is to display a box of the specified `size`.

```
(picmenu-select m &optional inside)
```

If the `picmenu` is not `flat`, its window should be destroyed following the selection using `menu-destroy`.

```
(picmenu-item-position m name &optional location)
```

```
(picmenu-delete-named-button m name:symbol)
```

This deletes a button from a displayed `picmenu`. The set of deleted buttons is reset to `nil` when the picmenu is drawn.

3.2 Barmenus

A `barmenu` displays a bar graph whose size can be adjusted using the mouse.

```
(barmenu-create maxval initval barwidth
            &optional title horizontal subtrackfn subtrackparms
            parentw x y perm flat color)
```

A value is selected by: `(barmenu-select m:barmenu &optional inside)`
If the `barmenu` is not `flat`, its window should be destroyed following the selection using `menu-destroy`.

The user must first click the mouse in the bar area; then the size of the displayed bar is adjusted as the user moves the mouse pointer. In addition, the `subtrackfn` is called with arguments of the size of the bar followed by the `subtrackparms`; this can be used, for example, to display a numeric value in addition to the bar size.

3.3 Menu Sets and Menu Conns

A `menu-set` is a set of multiple menus, picmenus, or barmenus that are simultaneously active within the same window. Menu-sets can be used to implement graphical user interfaces. A `menu-conns` is a menu-set that includes connections between menus; this can be used to implement interfaces that allow the user to construct a network from components.

The source file for menu-sets is the GLISP file `menu-set.lsp`; this file is translated as part of the file `drawtrans.lsp` in plain Lisp. Examples of the use of menu sets are given at the top of the file `menu-set.lsp`. In the following descriptions, `ms` is a `menu-set` and `mc` is a `menu-conns`.

`(menu-set-create w)` creates a menu set to be displayed in the window `w`.

`(menu-set-name symbol)` makes a `gensym` name that begins with `symbol`.

```
(menu-set-add-menu ms name:symbol sym title items
            &optional offset:vector)
```

This function adds a menu to a menu-set. `sym` is arbitrary information that is saved with the menu.

```
(menu-set-add-picmenu ms name sym title spec:picmenu-spec
            &optional offset:vector nobox)
```

```
(menu-set-add-component ms name &optional offset:vector)
```

This adds a component that has a `picmenu-spec` defined on the property list of `name`.

```
(menu-set-add-barmenu ms name sym barmenu title
            &optional offset:vector)
```

`(menu-set-draw ms)` draws all the menus.

```
(menu-set-select ms &optional redraw enabled)
```

`menu-set-select` gets a selection from a menu-set. If `redraw` is non-`nil`, the menu-set is drawn. `enabled` may be a list of names of menus that are enabled for selection. The result is `(selection menu-name)`, or `((x y) BACKGROUND button)` for a click outside any menu.

`(menu-conns-create ms)` creates a `menu-conns` from a `menu-set`.

```
(menu-conns-add-conn mc)
```

This function allows the user to select two ports from menus of the `menu-conns`. It then draws a line between the ports and adds the connection to the `connections` of the `menu-conns`.

```
(menu-conns-move mc)
```

This function allows the user to select a menu and move it. The `menu-set` and connections are redrawn afterwards.

```
(menu-conns-find-conn mc pt:vector)
```

This finds the connection selected by the point `pt`, if any. This is useful to allow the user to delete a connection:

```
(menu-conns-delete-conn mc conn)
```

```
(menu-conns-find-conns mc menuname port)
```

This returns all the connections from the specified `port` (selection) of the menu whose name is `menuname`.

4 Windows

```
(window-create width height &optional title parentw x y font)
```

`window-create` makes a new window of the specified `width` and `height`. `title`, if spec-

ified, becomes the displayed title of the window. If `parentw` is specified, it should be the `window-parent` property of an existing window, which becomes the parent window of the new window. `x` and `y` are the offset of the new window from the parent window. `font` is the font to be used for printing in the window; the default is given by `*window-default-font-name*`, initially `courier-bold-12`.

`(window-open w)` causes a window to be displayed on the screen.

`(window-close w)` removes the window from the display; it can be re-opened.

`(window-destroy w)`

`(window-moveto-xy w x y)`

`(window-geometry w)` queries X for the window geometry. The result is a list, `(x y width height border-width)` .

`(window-size w)` returns a list `(width height)` .
Note that the width and height are cached within the structure so that no call to X is needed to examine them. However, if the window is resized, it is necessary to call `(window-reset-geometry w)` to reset the local parameters to their correct values.

The following functions provide access to the parts of the `window` data structure; most applications will not need to use them.

```
(window-gcontext w)
(window-parent w)
(window-drawable-height w)
(window-drawable-width w)
(window-label w)
(window-font w)
(window-screen-height)
```

5 Drawing Functions

`(window-clear w)` clears the window to the background color.

`(window-force-output &optional w)`
Communication between the running program and X windows is done through a stream; actual drawing on the display is done asynchronously. `window-force-output` causes the current drawing commands, if any, to be sent to X. Without this, commands may be left in the stream buffer and may appear not to have been executed. The argument `w` is not used.

In all of the drawing functions, the `linewidth` argument is optional and defaults to `1`.

```
(window-draw-line w from:vector to:vector linewidth)
(window-draw-line-xy w x1 y1 x2 y2 &optional linewidth op)
```

`op` may be `xor` or `erase`.

```
(window-draw-arrow-xy w x1 y1 x2 y2 &optional linewidth size)
(window-draw-arrow2-xy w x1 y1 x2 y2 &optional linewidth size)
(window-draw-arrowhead-xy w x1 y1 x2 y2 &optional linewidth size)
```

These draw a line with an arrowhead at the second point, a line with an arrowhead at both points, or an arrowhead alone at the second point, respectively. `size` is the arrowhead size; the default is `(+ 20 (* linewidth 5))`.

```
(window-draw-box-xy w x y width height linewidth)
(window-xor-box-xy w x y width height linewidth)
(window-draw-box w offset:vector size:vector linewidth)
(window-draw-box-corners w x1 y1 x2 y2 linewidth)
```

where `(x1 y1)` and `(x2 y2)` are opposite corners.

```
(window-draw-rcbox-xy w x y width height radius linewidth)
```

draws a box with rounded corners.

```
(window-draw-arc-xy w x y radiusx radiusy anglea angleb linewidth)
```

`anglea` is the angle, in degrees, at which the arc is started. `angleb` is the angle, in degrees, that specifies the amount of arc to be drawn, counterclockwise from the starting position.

```
(window-draw-circle-xy w x y radius linewidth)
(window-draw-circle w center:vector radius linewidth)
(window-draw-ellipse-xy w x y radiusx radiusy linewidth)
(window-draw-dot-xy w x y)

(window-erase-area-xy w left bottom width height)
(window-erase-area w offset:vector size:vector)
(window-copy-area-xy w fromx fromy tox toy width height)
(window-invert-area w offset:vector size:vector)
(window-invert-area-xy w left bottom width height)

(window-printat-xy w s x y)
(window-printat w s at:vector)
(window-prettyprintat-xy w s x y)
(window-prettyprintat w s at:vector)
```

The argument `s` is printed at the specified position. `s` is stringified if necessary. Currently, the pretty-print versions are the same as the plain versions.

`(window-draw-border w)` draws a border just inside a window.

6 Fonts, Operations, Colors

```
(window-set-font w font)
```

The font symbols that are currently defined are `courier-bold-12`, `8x10`, and `9x15` . The global variable `*window-fonts*` contains correspondences between font symbols and font strings. A font string may also be specified instead of a font symbol.

```
(window-string-width w s)
(window-string-extents w s)
```

These give the width and the vertical size `(ascent descent)` in pixels of the specified string `s` using the font of the specified window. `s` is stringified if necessary.

Operations on a window other than direct drawing are performed by setting a condition for the window, performing the operation, and then unsetting the condition with `window-unset`. `window-reset` will reset a window to its "standard" setting; it is useful primarily for cases in which a program bug causes window settings to be in an undesired state.

```
(window-set-xor w)
(window-set-erase w)
(window-set-copy w)
(window-set-invert w)
(window-unset w)
(window-reset w)


(window-set-line-width w width)
(window-set-line-attr w width &optional line-style cap-style join-style)
(window-std-line-attr w)


(window-foreground w)
(window-set-foreground w fg-color)
(window-background w)
(window-set-background w bg-color)
```

6.1 Color

The color of the foreground (things that are drawn, such as lines or characters) is set by:

```
(window-set-color w rgb &optional background)
```

`rgb` is a list `(red green blue)` of 16-bit unsigned integers in the range `0` to `65535`. `background` is non-`nil` to set the background color rather than the foreground color.

Colors are a scarce resource; there is only a finite number of available colors, such as 256 colors. If you only use a small, fixed set of colors, the finite set of colors will not be a

problem. However, if you create a lot of colors that are used only briefly, it will be necessary to release them after they are no longer needed. `window-set-color` will leave the global variable `*window-xcolor*` set to an integer value that denotes an X color; this value should be saved and used as the argument to `window-free-color` to release the color after it is no longer needed.

```
(window-free-color w &optional xcolor)
```

`window-free-color` frees either the last color used, as given by `*window-xcolor*`, or the specified color.

6.2 Character Input

Characters can be input within a window by the call:

```
(window-input-string w str x y &optional size)
```

`window-input-string` will print the initial string `str`, if non-`nil`, at the specified position in the window; `str`, if not modified by the user, will also be the initial part of the result. A caret is displayed showing the location of the next input character. Characters are echoed as they are typed; backspacing erases characters, including those from the initial string `str`. An area of width `size` (default 100) is erased to the right of the initial caret.

6.3 Emacs-like Editing

`window-edit` allows editing of text using an Emacs-subset editor. Only a few simple Emacs commands are implemented.

```
(window-edit w x y width height &optional strings boxflg scroll endp)
```

`x y width height` specify the offset and size of the editing area; it is a good idea to draw a box around this area first. `strings` is an initial list of strings; the return value is a list of strings. `scroll` is number of lines to scroll down before displaying text, or `T` to have one line only and terminate on return. `endp` is `T` to begin editing at the end of the first line. Example:

```
(window-draw-box-xy myw 48 48 204 204)
(window-edit myw 50 50 200 200 '("Now is the time" "for all" "good"))
```

7 Mouse Interaction

```
(window-get-point w)
(window-get-crosshairs w)
(window-get-cross w)
```

These functions get a point position by mouse click; they return `(x y)` .

The following function gets a point position by mouse click. It returns `(button (x y))` where `button` is `1` for the left button, `2` for middle, `3` for right.

```
(window-get-click w)
```

The following function gets a point position by mouse click within a specified region. It returns `(button (x y))` or `NIL` if the mouse leaves the region. If `boxflg` is `t`, a box will be drawn outside the region while the mouse is being tracked.

```
(window-track-mouse-in-region w x y sizex sizey &optional boxflg)
```

The following functions get a point position indicated by drawing a line from a specified origin position to the cursor position; they return `(x y)` at the cursor position when a mouse button is clicked. The `latex` version restricts the slope of the line to be a slope that LaTeX can draw; if `flg` is non-`nil`, the slope is restricted to be a LaTeX `vector` slope.

```
(window-get-line-position w orgx orgy)
(window-get-latex-position w orgx orgy flg)
```

The following function gets a position by moving a "ghost" icon, defined by the icon drawing function `fn`. This allows exact positioning of an object by the user.

```
(window-get-icon-position w fn args &optional (dx 0) (dy 0))
```

The function `fn` has arguments `(fn w x y . args)` , where `x` and `y` are the offset within the window `w` at which the icon is to be drawn, and `args` is a list of arbitrary arguments, e.g., the size of the icon, that are passed through to the drawing function. The icon is drawn in `xor` mode, so it must be drawn using only "plain" drawing functions, without resetting window attributes. The returned value is `(x y)` at the cursor position when a button is clicked. `dx` and `dy`, if specified, are offsets of `x` and `y` from the cursor position.

The following function gets a position by moving a "ghost" box icon.

```
(window-get-box-position w width height &optional (dx 0) (dy 0))
```

By default, the lower-left corner of the box is placed at the cursor position; `dx` and `dy` may be used to offset the box from the cursor, e.g., to move the box by a different corner. The returned value is `(x y)` at the cursor position when a button is clicked.

The following function gets coordinates of a box of arbitrary size and position.

```
(window-get-region w)
```

The user first clicks for one corner of the box, moves the mouse and clicks again for the opposite corner, then moves the box into the desired position. The returned value is `((x y) (width height))`, where `(x y)` is the lower-left corner of the box.

The following function gets the size of a box by mouse selection, echoing the size in pixels below the box. `offsety` should be at least `30` to leave room to display the size of the box.

```
(window-get-box-size w offsetx offsety)
```

The following function adjusts one side of a box.

```
(window-adjust-box-side w x y width height side)
```

`side` specifies the side of the box to be adjusted: `left`, `right`, `top`, or `bottom`. The result is `((x y) (width height))` for the resulting box.

```
(window-get-circle w &optional center:vector)
(window-get-ellipse w &optional center:vector)
```

These functions interactively get a circle or ellipse. For an ellipse, a circle is gotten first for the horizontal size; then the vertical size of the ellipse is adjusted. `window-get-circle` returns `((x y) radius)`. `window-get-ellipse` returns `((x y) (xradius yradius))`.

`window-track-mouse` is the basic function for following the mouse and performing some action as it moves. This function is used in the implementation of menus and the mouse-interaction functions described in this section.

```
(window-track-mouse w fn &optional outflg)
```

Each time the mouse position changes or a mouse button is pressed, the function `fn` is called with arguments `(x y code)` where `x` and `y` are the cursor position, `code` is a button code (`0` if no button, `1` for the left button, `2` for the middle button, or `3` for the right button). `window-track-mouse` continues to track the mouse until `fn` returns a value other than `nil`, at which time `window-track-mouse` returns that value. Usually, it is a good idea for `fn` to return a value other than `nil` upon a mouse click. If the argument `outflg` is non-`nil`, the function `fn` will be called for button clicks outside the window `w`; note, however, that such clicks will not be seen if the containing window intercepts them, so that this feature will work only if the window `w` is inside another Lisp user window.

8 Miscellaneous Functions

`(stringify x)` makes its argument into a string.

`(window-destroy-selected-window)` waits 3 seconds, then destroys the window containing the mouse cursor. This function should be used with care; it can destroy a non-user window, causing processes associated with the window to be destroyed. It is useful

primarily in debugging, to get rid of a window that is left on the screen due to an error.

9 Examples

Several interactive programs using this software for their graphical interface can be found at `http://www.cs.utexas.edu/users/novak/` under the heading Software Demos.

10 Web Interface

This software allows a Lisp program to be used interactively within a web page. There are two approaches, either using an X server on the computer of the person viewing the web page, or using WeirdX, a Java program that emulates an X server. Details can be found at: `http://www.cs.utexas.edu/users/novak/dwindow.html`

11 Files

`dec.copyright`	Copyright and license for DEC/MIT files
`draw.lsp`	GLISP source code for interactive drawing utility
`drawtrans.lsp`	`draw.lsp` translated into plain Lisp
`draw-gates.lsp`	Code to draw `nand` gates etc.
`dwdoc.tex`	LaTeX source for this document
`dwexports.lsp`	exported symbols
`dwimportsb.lsp`	imported symbols
`dwindow.lsp`	GLISP source code for `dwindow` functions
`dwtest.lsp`	Examples of use of `dwindow` functions
`dwtrans.lsp`	`dwindow.lsp` translated into plain Lisp
`editors.lsp`	Editors for colors etc.
`editorstrans.lsp`	translation of `editors.lsp`
`gnu.license`	GNU General Public License
`ice-cream.lsp`	Drawing of an ice cream cone made with `draw`
`lispserver.lsp`	Example web demo: a Lisp server
`lispservertrans.lsp`	translation of `lispserver.lsp`
`menu-set.lsp`	GLISP source code for menu-set functions
`menu-settrans.lsp`	translation of `menu-set.lsp`
`pcalc.lsp`	Pocket calculator implemented as a `picmenu`

12 Data Types

```
(window (listobject  (parent          drawable)
                     (gcontext        anything)
                     (drawable-height integer)
                     (drawable-width  integer)
                     (label           string)
                     (font            anything) )

(menu (listobject (menu-window     window)
                  (flat            boolean)
                  (parent-window   drawable)
                  (parent-offset-x integer)
                  (parent-offset-y integer)
                  (picture-width   integer)
                  (picture-height  integer)
                  (title           string)
                  (permanent       boolean)
                  (menu-font       symbol)
                  (item-width      integer)
                  (item-height     integer)
                  (items           (listof symbol)) )

(picmenu (listobject (menu-window     window)
                     (flat            boolean)
                     (parent-window   drawable)
                     (parent-offset-x integer)
                     (parent-offset-y integer)
                     (picture-width   integer)
                     (picture-height  integer)
                     (title           string)
                     (permanent       boolean)
                     (spec            (transparent picmenu-spec))
                     (boxflg          boolean)
                     (deleted-buttons (listof symbol)) )

(picmenu-spec (listobject (drawing-width   integer)
                          (drawing-height  integer)
                          (buttons         (listof picmenu-button))
                          (dotflg          boolean)
                          (drawfn          anything)
                          (menu-font       symbol) ))
```

```
(picmenu-button (list (buttonname     symbol)
                      (offset         vector)
                      (size           vector)
                      (highlightfn    anything)
                      (unhighlightfn  anything))

(barmenu (listobject (menu-window     window)
                     (flat            boolean)
                     (parent-window   drawable)
                     (parent-offset-x integer)
                     (parent-offset-y integer)
                     (picture-width   integer)
                     (picture-height  integer)
                     (title           string)
                     (permanent       boolean)
                     (color           rgb)
                     (value           integer)
                     (maxval          integer)
                     (barwidth        integer)
                     (horizontal      boolean)
                     (subtrackfn      anything)
                     (subtrackparms   (listof anything)))
```

13 Copyright

The following copyright notice applies to the portions of the software that were adapted from X Consortium software:

```
;;************************************************************
;;Copyright 1987 by Digital Equipment Corporation, Maynard, Massachusetts,
;;and the Massachusetts Institute of Technology, Cambridge, Massachusetts.

;;                        All Rights Reserved

;;Permission to use, copy, modify, and distribute this software and its
;;documentation for any purpose and without fee is hereby granted,
;;provided that the above copyright notice appear in all copies and that
;;both that copyright notice and this permission notice appear in
;;supporting documentation, and that the names of Digital or MIT not be
;;used in advertising or publicity pertaining to distribution of the
;;software without specific, written prior permission.

;;DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
;;ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL
;;DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR
;;ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
;;WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
;;ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS
;;SOFTWARE.

;;************************************************************************
```