4.23 Status Meters

A status meter is like a progress bar. It has either a vertical or a horizontal orientation. A status meter is capable of displaying five distinct quantities simultaneously.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_STATUS_METER`
Default instance name:	`"statusMeter"`
Default Class name:	`"StatusMeter"`

Widget Interface Routines

`void  EZ_SetStatusMeterRange(EZ_Widget *widget, float f, float t);`

set the range of a status meter.

`void  EZ_SetStatusMeterValue(EZ_Widget *widget, float value);`

set the value of a status meter.

`float EZ_GetStatusMeterValue(EZ_Widget *widget);`

return the status value.

`void  EZ_StopStatusMeter(EZ_Widget *widget);`

stop a status meter.

`void  EZ_StartStatusMeter(EZ_Widget *widget);`

resume a status meter.

`void  EZ_SetStatusMeterForeground(EZ_Widget *widget, char *clr);`

set the foreground color.

`void  EZ_SetStatusMeterBackground(EZ_Widget *widget, char *clr);`

set the background color.

`void  EZ_SetStatusMeterRangeN(EZ_Widget *widget, int idx, float f, float t);`

set the rang of the n*th* status variable.

`void  EZ_SetStatusMeterForegroundN(EZ_Widget *widget, int idx, char *clr);`

set the foreground color to display the n*th* status variable.

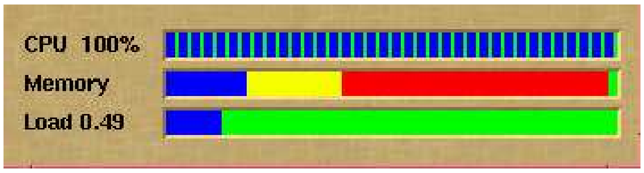
`void  EZ_SetStatusMeterValueN(EZ_Widget *widget, int idx, float value);`

set the value of th n*th* status variable.

`float EZ_GetStatusMeterValueN(EZ_Widget *widget, int idx);`

return the value of the n*th* status variable

Screenshot



Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"StatusMeter"`
EZ_NAME	name	`"statusmeter"`

EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	

EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	

EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	

EZ_BORDER_WIDTH	borderWidth	`0`
EZ_BORDER_TYPE	borderType	`none`
EZ_FOCUS_PAD	focusPad	`0`

EZ_PADX	padx	`0`
EZ_PADY	pady	`0`
EZ_IPADX	iPadx	`0`
EZ_IPADY	iPady	`0`

EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`

EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`

EZ_FOREGROUND	foreground	`"black"`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`

EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`

EZ_SMETER_VALUE	statusMeterValue	`0.0`
EZ_SMETER_RANGE	statusMeterRange	`0.0 0.0`
EZ_SMETER_STOPED	statusMeterStoped	`false`
EZ_SMETER_BACKGROUND	statusMeterBackground	`"gray74"`
EZ_SMETER_FOREGROUND	statusMeterForeground	`"blue"`
EZ_SMETER_FOREGROUND_N	statusMeterForegroundN	
EZ_SMETER_SHOW_VALUE	statusMeterShowValue	`false`
EZ_SMETER_VALUE_N	statusMeterValueN	
EZ_SMETER_RANGE_N	statusMeterRangeN	
EZ_SMETER_STYLE	statusMeterStyle	`0`
EZ_SMETER_ORIENTATION	statusMeterOrientation	`horizontal`

4.24 Menubar

A menubar is a window to hold a set of menubutton like items (no window labels). When button 1 is pressed over an item, it posts the popup menu assigned to the item. Then the user can slide the mouse down over the menu with button still pressed and release over a desired item. When the button is released over a menu item, the menu is unposted and if the menu item has any callbacks, they are invoked first. In the mean time, the selected menu entry and its return identifier are recorded in the most recently posted menu (parent of the selected item). If the menu has callbacks, they are invoked next.

A menubar may have an optional docking handle.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_MENU_BAR`
Default instance name:	`"menuBar"`
Default Class name:	`"MenuBar"`

Widget Interface Routines

```
EZ_Widget *EZ_MenuBarAddItemAndMenu(EZ_Widget *mbar, char *label,
                                    int uline, EZ_Widget *menu);
```

append an item to a menubar.

```
EZ_MenuBarSetItemMenu(EZ_Widget *mbar, char *label, EZ_Widget *menu);
```

associate a menu to an item on a menubar.

```
void EZ_MenuBarInsertItemAndMenu(EZ_Widget *mbar, char *label, int ul,
                     EZ_Widget *menu,  char *refItem, int where);
```

insert an item near the specified reference item.

```
void  EZ_MenuBarDeleteItem(EZ_Widget *mbar, char *label);
```

remove an item from a menubar.

```
void EZ_MenuBarDeleteItemAndMenu(EZ_Widget *mbar, char *label);
```

delete an item and destroy its associated menu.

```
EZ_Widget   *EZ_MenuBarFindItem( EZ_Widget *mbar, char *name);
```

find and return the specified item.

```
EZ_Widget *EZ_MenuBarFindItemAndMenu( EZ_Widget *mbar, char *name,
                                      EZ_Widget **menuReturn);
```

find and return the return an item. Return its associated menu also.

```
EZ_Widget   *EZ_MenuBarFindItemAtXY( EZ_Widget *mbar, int x, int y);
```

return the item displayed at the specified location.

```
EZ_Widget *EZ_MenuBarFindItemAndMenuAtXY(EZ_Widget *mbar, int x, int y,
                                         EZ_Widget **menuRet);
```

return the item and its associated menu displayed at the specified location.

```
int EZ_OnFrameDragHandle(EZ_Widget *mbar int x, int y);
```

return 1 if point (x,y) is on the drag handle.

```
void   EZ_HighlightFrameDragHandle(EZ_Widget *mbar int onoff);
```

turn the highlight on the drag handle on or off.

```
int  EZ_DetachFrame(EZ_Widget *mbar int x, int y);
```

detach a menubar from its parent.

```
int EZ_DockFrame(EZ_Widget *mbar int force);
```

try dock a detached menubar.

```
int EZ_XYIsInCorner(int x,int y,int w,int h);
```

returns 1 if point (x,y) is on the corner of a detached menubar.

```
int EZ_FrameIsDetached(EZ_Widget *widget);
```

returns 1 if frame/menubar is detached.

Screenshot

Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"MenuBar"`
EZ_NAME	name	`"menuBar"`
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	
EZ_BORDER_WIDTH	borderWidth	`2`
EZ_BORDER_TYPE	borderType	`raised`
EZ_FOCUS_PAD	focusPad	`0`
EZ_PADX	padx	`8`
EZ_PADY	pady	`2`
EZ_IPADX	iPadx	`8`
EZ_IPADY	iPady	`0`
EZ_EXPAND	expand	`true`
EZ_PROPAGATE	propagate	`true`
EZ_ORIENTATION	orientation	`horizontalLeft`
EZ_FILL_MODE	fillMode	`fillHorizontally`
EZ_SIDE	side	`center`
EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`
EZ_CALLBACK	N/A	`null null`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`
EZ_FOREGROUND	foreground	`"black"`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`
EZ_HIGHLIGHT_FOREGROUND	highlightForeground	
EZ_HIGHLIGHT_BACKGROUND	highlightBackground	
EZ_DRAG_HANDLE	dragHandle	`none`
EZ_DOCKABLE	dockable	`false`
EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`
EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`

4.25 Raw XWindow

This widget gives you a native X window. You'll have to write a rendering procedure and an event handler for it to be useful. This widget is also used internally to hook up user defined primitive widgets.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_RAW_XWINDOW`
Default instance name:	`"rawXWindow"`
Default Class name:	`"RawXWindow"`

Widget Interface Routines

```
void EZ_SetupRawXWinWidget(EZ_Widget *widget,
           void (*ComputeSize)( EZ_Widget *widget, int *w, int *h),
           void (*DrawWidget) (EZ_Widget *widget),
           void (*FreeData)   (EZ_Widget *widget),
           void (*EventHandle)(EZ_Widget *widget, XEvent *event) );
```

setup a raw xwindow widget.

```
int   EZ_RegisterPrimitiveWidget(
            char  *iname, char *cname,
            void  (*Setup) (EZ_Widget *widget),
            void  (*ComputeSize)( EZ_Widget *widget, int *w, int *h),
            void  (*DrawWidget) (EZ_Widget *widget),
            void  (*FreeData)   (EZ_Widget *widget),
            void  (*EventHandle)(EZ_Widget *widget, XEvent *event)) );
```

register your own widget.

Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"RawXWindow"`
EZ_NAME	name	`"rawXWindow"`
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_BORDER_WIDTH	borderWidth	`0`
EZ_BORDER_TYPE	borderType	`none`
EZ_FOCUS_PAD	focusPad	`0`
EZ_PADX	padx	`0`
EZ_PADY	pady	`0`
EZ_IPADX	iPadx	`0`
EZ_IPADY	iPady	`0`
EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`true`
EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`
EZ_CALLBACK	N/A	`null null`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`
EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`

4.26 FreeLabels

A freeLabel is a label whose position is not managed by the geometry manager. FreeLabel also responds to button inputs. Button1 press-drag-release moves a FreeLabel around and double Button1 click involes its callbacks. It can also serves as a movable container widget.

FreeLabels are used to implement icons.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_FREE_LABEL`
Default instance name:	`"freeLabel"`
Default Class name:	`"FreeLabel"`

Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"FreeLabel"`
EZ_NAME	name	`"freeLabel"`
EZ_X	x	`0`
EZ_Y	y	`0`
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	
EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	
EZ_BORDER_WIDTH	borderWidth	`0` or `2`
EZ_BORDER_TYPE	borderType	`none` or `raised`
EZ_FOCUS_PAD	focusPad	`0`
EZ_PADX	padx	`8`
EZ_PADY	pady	`6`
EZ_IPADX	iPadx	`2`
EZ_IPADY	iPady	`2`
EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`false`
EZ_TRANSIENT	transient	`false`
EZ_ORIENTATION	orientation	`horizontal`
EZ_FILL_MODE	fillMode	`fillNone`
EZ_SIDE	side	`center`
EZ_FONT_ID	fontID	`3`
EZ_FONT_NAME	fontName	`"*Bold*"`
EZ_LABEL_STRING	labelString	`null`
EZ_LABEL_POSITION	labelPosition	`center`
EZ_JUSTIFICATION	justification	`left`
EZ_TEXT_LINE_LENGTH	textLineLength	`30`
EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`
EZ_CALLBACK	N/A	`null null`
EZ_MOTION_CALLBACK	N/A	`null null`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`

EZ_PIXMAP_FILE	pixmapFile	`null`
EZ_IMAGE_FILE	imageFile	`null`
EZ_BITMAP_DATA	bitmapData	`null`
EZ_PIXMAP_DATA	pixmapData	`null`
EZ_X_PIXMAP	xPixmap	`none`
EZ_LABELPIXMAP	labelPixmap	`null`
EZ_RAW_RGB_DATA	rawRGBData	`null`
EZ_SHAPED_WINDOW	shapedWindow	`false`
EZ_SHAPE_FILE	shapeFile	`NULL`
EZ_SHAPE_PIXMAP	N/A	`NULL`

EZ_FOREGROUND	foreground	`"black"`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`

EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`

EZ_LABELED_ICON	labeledIcon	`0`
EZ_MOVABLE	movable	`false`
EZ_INTERIOR_BORDER	interiorBorder	`"0 none"`

4.27 No Window Label

No-window-labels are used to build visually complex labels for button type widgets. It is also used in the implementation of the menubar widget.

A no-window-label is visually the same as a label widget. The major difference is that a no-window-label does not have its own window and hence (1) it cannot be used as toplevel widgets and (2) it does not handle any events.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_NW_LABEL`
Default instance name:	`"nWLabel"`
Default Class name:	`"NWLabel"`

Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"NWLabel"`
EZ_NAME	name	`"nWLabel"`
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	
EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	
EZ_BORDER_WIDTH	borderWidth	`0`
EZ_BORDER_TYPE	borderType	`none`
EZ_FOCUS_PAD	focusPad	`0`
EZ_PADX	padx	`0`
EZ_PADY	pady	`0`
EZ_IPADX	iPadx	`8`
EZ_IPADY	iPady	`6`
EZ_FONT_ID	fontID	`3`
EZ_FONT_NAME	fontName	`"*Bold*"`
EZ_LABEL_STRING	labelString	`null`
EZ_LABEL_POSITION	labelPosition	`center`
EZ_JUSTIFICATION	justification	`left`
EZ_TEXT_LINE_LENGTH	textLineLength	`30`
EZ_ARROW_LABEL	arrowLabel	`0`
EZ_PIXMAP_FILE	pixmapFile	`null`
EZ_IMAGE_FILE	imageFile	`null`
EZ_BITMAP_DATA	bitmapData	`null`
EZ_PIXMAP_DATA	pixmapData	`null`
EZ_X_PIXMAP	xPixmap	`none`
EZ_LABELPIXMAP	labelPixmap	`null`
EZ_RAW_RGB_DATA	rawRGBData	`null`
EZ_SHAPED_WINDOW	shapedWindow	`false`
EZ_FOREGROUND	foreground	`"black"`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`
EZ_HIGHLIGHT_FOREGROUND	highlightForeground	
EZ_HIGHLIGHT_BACKGROUND	highlightBackground	
EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

4.28 The 3D Canvas Widget

The 3D-canvas widget realizes the EZ graphics library. To each 3DCanvas widget, there is an associated data structure called *the graphics context* which contains the entire state attributes of the graphics library. The graphics context is essentially private to the 3D-canvas widget and hence multiple 3Dcanvases is allowed in an application.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_3D_CANVAS`
Default instance name:	`"threeDCanvas"`
Default Class name:	`"ThreeDCanvas"`

Widget Interface Routines

`void EZ_Set3DCanvas( EZ_Widget *canvas);`

make a canvas the current active one (GL context switch).

`void EZ_Configure3DCanvas( EZ_Widget *widget);`

update the state variables of the graphics context associated to a 3d canvas.

`void EZ_GLConfig(void);`

update the state variables of the current graphics context.

`void EZ_Get3DCanvasSize(EZ_Widget *widget, int *w, int *h);`

return the dimension of a 3d canvas.

`XImage *EZ_Read3DCanvas2XImage(EZ_Widget *widget);`

save the graphics window to an XImage.

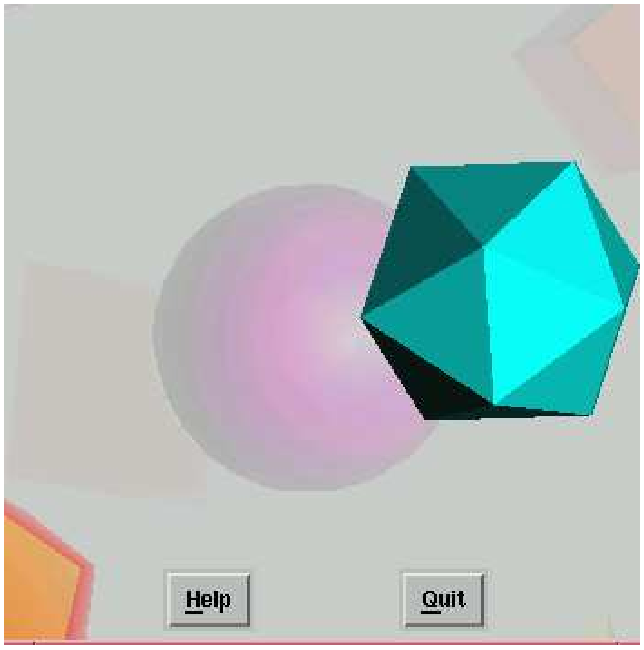
`void EZ_Save3DCanvas2PPMImage(EZ_Widget *widget, char *fname);`

save the grahics window to disk as a PPM image.

`void EZ_Save3DCanvas2PS(EZ_Widget *widget, char *fname);`

save the graphics window to postscript.

Screenshot



Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`ThreeDCanvas`
EZ_NAME	name	`threeDCanvas`
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	
EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	
EZ_PROPAGATE	propagate	`true`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`
EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`
EZ_BACKING_STORE	backingStore	`true`
EZ_RESERVE_MENU_BUTTON	reserveMenuButton	`false`

4.29 Terminal Emulator

EZwgl implements a VT100 terminal emulator. The emulator code is modified from the "xiterm" source code. Thanks to the authors of xiterm.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_TERM`
Default instance name:	`"term"`
Default Class name:	`"Term"`
Symbolic Widget type:	`EZ_WIDGET_ITERM`
Default instance name:	`"iTerm"`
Default Class name:	`"ITerm"`

Subwidgets Info

Subwidgets of Text are listed below.

```
       ( iTerm
term-->(
       ( verticalScrollbar
```

Widget Interface Routines

```
void EZ_GetTermComponents(EZ_Widget *widget, EZ_Widget **iterm,
                              EZ_Widget **vscr );
```

return the components of a terminal emulator.

```
void  void EZ_TermStartCommand(EZ_Widget *term, char **args);
```

run the specified program in the terminal emulator window.

```
void  void EZ_TermStopCommand(EZ_Widget *term, int sig);
```

send the specified signal to the process running in the terminal emulator.

```
void   EZ_TermSetExitHandler(EZ_Widget *ezterm,
                        EZ_CallBack cb,  void *data);
```

Specifies a procedure to be invoked when the child process running in the terminal emulator exits.

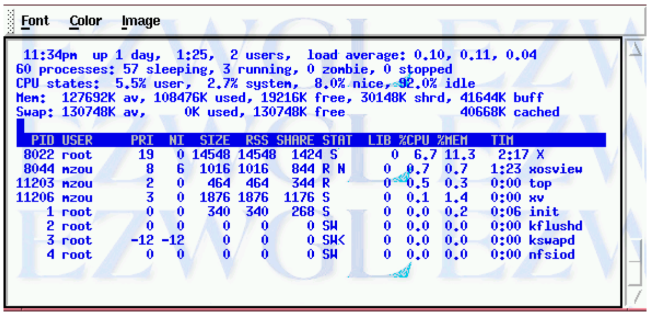
```
void EZ_TermTouchScreen(EZ_Widget *term);
```

This procedure changes the text attributes currently displayed in the emulator window, and cause the screen to be refreshed completely.

```
void EZ_TermPowerOn(EZ_Widget *term);
```

This procedure completely resets the display screen.

Screenshot



Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"Term"`
		or `"ITerm"`
EZ_NAME	name	`"term"`
		or `"iTerm"`
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	
EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	
EZ_BORDER_WIDTH	borderWidth	`0` or `2`
EZ_BORDER_TYPE	borderType	`none` or `sunken`
EZ_FOCUS_PAD	focusPad	`0` or `2`
EZ_PADX	padx	`8` or `0`
EZ_PADY	pady	`6` or `0`
EZ_IPADX	iPadx	`0`
EZ_IPADY	iPady	`0`
EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`true`
EZ_TRANSIENT	transient	`false`
EZ_FONT_ID	fontID	`3`
EZ_FONT_NAME	fontName	`"*fixed*"`
EZ_CURSOR	cursor	`"XC_xterm"`
EZ_BUBBLE_STRING	bubbleString	`null`
EZ_CALLBACK	N/A	`null null`
EZ_MOTION_CALLBACK	N/A	`null null`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`
EZ_FOREGROUND	foreground	`"black"`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`
EZ_BG_IMAGE_FILE_B	bgImageFileB	`null`
EZ_BG_PIXMAP_B	N/A	`none`
EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`
EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`
EZ_OPTIONAL_VSCROLLBAR	optionalVScrollbar	`true`
EZ_SCROLLBAR_WIDTH	scrollbarWidth	`10`
EZ_SCROLLBAR_BORDER_WIDTH	scrollbarBorderWidth	`2`
EZ_VSCROLL_INCREMENT	vScrollIncrement	`1`
EZ_TEXT_BACKGROUND	textBackground	
EZ_TERM_COLOR_N	termColorN	
EZ_TERM_CURSOR_COLOR	termCursorColor	
EZ_TERM_SAVE_LINES	termSaveLines	`64`
EZ_TERM_REVERSE_VIDEO	termReverseVideo	`false`

4.30 Ruler

A ruler is controled by the following parameters:

- `tick unit`, the number of pixels between tick marks (default is 5).
- `tick scale`, length represented by 10 tick units (default is 10).
- `tick length`, the length of the longest tick marker (default is 16).
- `offset`, initial ruler offset (default is 0).
- `position`, location of the ruler. Default value is `EZ_TOP` for horizontal rulers and `EZ_LEFT` for vertical rulers.
- `display value`, a flag indicates whether or not to display numerical tick marks.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_HORIZONTAL_RULER`
Default instance name:	`"horizontalRuler"`
Default Class name:	`"HorizontalRuler"`

Symbolic Widget type:	`EZ_WIDGET_VERTICAL_RULER`
Default instance name:	`"verticalRuler"`
Default Class name:	`"VerticalRuler"`

Widget Interface Routines

```
void EZ_WidgetNotifyRuler(EZ_Widget *ruler, int x, int y,
                              EZ_Widget *sourceWidget );
```

Notify ruler the position of the mouse pointer, relative to the window of `srcWidget`.

```
void EZ_XWindowNotifyRuler(EZ_Widget *ruler, int x, int y,
                              Window window);
```

Notify ruler the position of the mouse pointer, relative to `window`.

```
void EZ_WidgetMarkRuler(EZ_Widget *ruler, int x, int y,
                              EZ_Widget *sourceWidget );
```

Set one of the two marks on the ruler.

```
void EZ_XWindowMarkRuler(EZ_Widget *ruler, int x, int y,
                              Window window);
```

Set one of the two marks on the ruler.

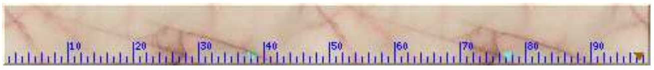
```
void EZ_SetRulerOffset(EZ_Widget *ruler, int offset);
```

Sets ruler offset in pixels. The origin of the ruler window will be treated as `-offset` pixels from the imaginary ruler origin.

```
void  void EZ_SetupRuler(EZ_Widget *ruler, int tickUnit, int tickLength,
                              int numScale, int offset)
```

Sets all control parameters. `tickUnit` is distance between tick marks; `tickLength` is the length of the longest tick marks; `numScale` represents the value corresponds to 10 tick units and `offset` is the ruler offset.

Screenshot



Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"HorizontalRuler"` or `"VerticalRuler"`
EZ_NAME	name	`"horizontalRuler"` or `"verticalRuler"`

EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	

EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	

EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	

EZ_BORDER_WIDTH	borderWidth	`0` or `2`
EZ_BORDER_TYPE	borderType	`none` or `sunken`

EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`true`
EZ_TRANSIENT	transient	`false`

EZ_FONT_ID	fontID	`5`
EZ_FONT_NAME	fontName	`"5x7"`

EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`

EZ_FOREGROUND	foreground	`"black"`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`

EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

EZ_RULER_DISPLAY_VALUE	rulerDisplayValue	`True`
EZ_RULER_TICK_UNIT	rulerTickUnit	`5`
EZ_RULER_TICK_LENGTH	rulerTickLength	`16`
EZ_RULER_TICK_SCALE	rulerTickScale	`10`
EZ_RULER_TICK_OFFSET	rulerTickOffset	`0`
EZ_RULER_POSITION	rulerposition	`top`
		or `"left"`

4.31 Scroll Button

A scroll button displays one or two arrows. When mouse button 1 (left button) is pressed over an arrow, its callbacks are invoked. The callbacks will be executed continuously as long as the mouse button 1 is pressed. A scroll button has one state variable, the number of times the callbacks has been executed since last button press.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_SCROLL_BUTTON`
Default instance name:	`"scrollButton"`
Default Class name:	`"ScrollButton"`

Widget Interface Routines

`int EZ_GetScrollButtonScrollCount(EZ_Widget *scrollButton);`

returns the current signed scroll count.

Screenshot

Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"ScrollButton"`
EZ_NAME	name	`"scrollButton"`
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	
EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	
EZ_BORDER_WIDTH	borderWidth	`0 or 2`
EZ_BORDER_TYPE	borderType	`none or sunken`
EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`true`
EZ_TRANSIENT	transient	`false`
EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`
EZ_FOREGROUND	foreground	`"black"`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`
EZ_SHAPED_WINDOW	shapedWindow	`false`
EZ_SHAPE_FILE	shapeFile	`NULL`
EZ_SHAPE_PIXMAP	N/A	`NULL`

EZ_BG_IMAGE_FILE_B	bgImageFileB	`null`
EZ_BG_PIXMAP_B	N/A	`none`
EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

EZ_SCROLL_TYPE	scrollType	`int`

4.32 Spin Button

A spin button is a composite widget, it consists of an entry and a scroll button. A spin button is used to cycle through a series of string type inputs. The value is controled by a spin function of type

```
char *(func)(int old_idx, int new_idx, void *clientData);
```

When the left mouse button is pressed over the arrow button, the spin function `func` will be called to compute the next choice. The value will then be displayed in the entry widget. Here is an example.

```
static char *months[]={"January", "Febrary", "March", "April", "May",
                       "June", "July", "August", "September", "October",
                       "November", "December"};
static char *spinFunc(int last, int current, void *data)
{
  current %= 12;
  if(current < 0) current += 12;
  return(months[current]);
}

main(int ac, char **av)
{
  EZ_Widget *spin;

  EZ_Initialize(ac, av, 0);

  spin  = EZ_CreateWidget(EZ_WIDGET_SPIN_BUTTON,   NULL,
                          EZ_SPIN_VALUE,           3,   "April",
                          EZ_SPIN_FUNCTION,        nextF, NULL,
                          0);
  EZ_DisplayWidget(spin);
  EZ_EventMainLoop();
}
```

Widget Info

Symbolic Widget type:	`EZ_WIDGET_SPIN_BUTTON`
Default instance name:	`"spinButton"`
Default Class name:	`"SpinButton"`

Widget Interface Routines

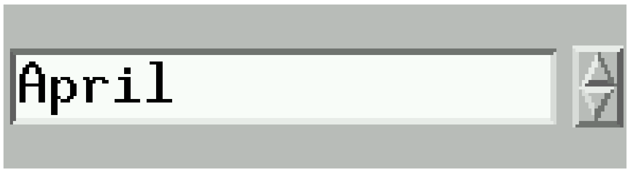
`char  *EZ_GetSpinButtonValue(EZ_Widget *spinButton);`

return the spin button selection.

`void  EZ_SetupSpinButton(EZ_Widget *spinBtn, char *value,`
`          char *(nextValue)(int,int void *));`

setup a spin button.

Screenshot



Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	"Locator"
EZ_NAME	name	"locator"
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	

EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	

EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	

EZ_BORDER_WIDTH	borderWidth	`2`
EZ_BORDER_TYPE	borderType	`sunken`
EZ_FOCUS_PAD	focusPad	`2`

EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`true`
EZ_TRANSIENT	transient	`false`

EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`

EZ_MOTION_CALLBACK	N/A	`null null`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`

EZ_FOREGROUND	foreground	`"black"`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`
EZ_SHAPED_WINDOW	shapedWindow	`false`
EZ_SHAPE_FILE	shapeFile	`NULL`
EZ_SHAPE_PIXMAP	N/A	`NULL`

EZ_BG_IMAGE_FILE_B	bgImageFileB	`null`
EZ_BG_PIXMAP_B	N/A	`none`
EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`

EZ_SPIN_VALUE	spinValue	`0, NULL`
EZ_SPIN_FUNCTION	N/A	`NULL,NULL`

4.33 Locator

Widget Info

Symbolic Widget type:	`EZ_WIDGET_LOCATOR`
Default instance name:	`"locator"`
Default Class name:	`"Locator"`

Widget Interface Routines

```
void  EZ_GetLocatorXY(EZ_Widget *locator, float *x, float *y);
```

returns the locator position.

```
void  EZ_SetLocatorXY(EZ_Widget *locator, float x, float y);
```

set the locator.

Screenshot

Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"Locator"`
EZ_NAME	name	`"locator"`
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	
EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	
EZ_BORDER_WIDTH	borderWidth	2
EZ_BORDER_TYPE	borderType	`sunken`
EZ_FOCUS_PAD	focusPad	2
EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`true`
EZ_TRANSIENT	transient	`false`

EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`

EZ_MOTION_CALLBACK	N/A	`null null`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`

EZ_FOREGROUND	foreground	`"black"`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`

EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`

EZ_LOCATOR_XY	locatorXY	`0.5,0.5`

4.34 Dial

Widget Info

Symbolic Widget type:	`EZ_WIDGET_DIAL`
Default instance name:	`"dial"`
Default Class name:	`"Dial"`

Widget Interface Routines

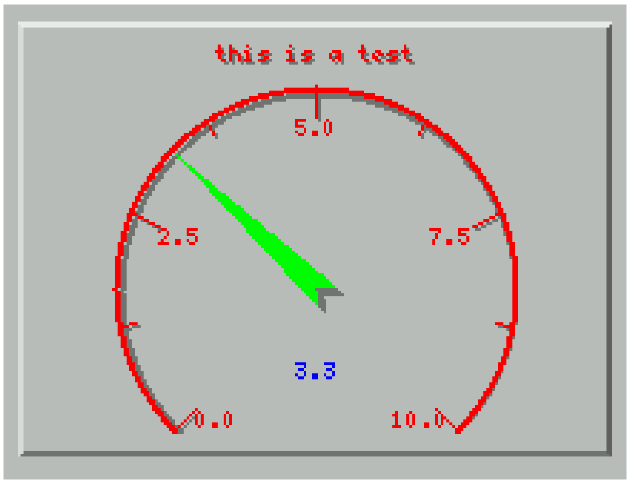
```
void   EZ_SetDialValue(EZ_Widget *widget, float value);
```

set dial value.

```
float  EZ_GetDialValue(EZ_Widget *widget);
```

retrieve dial value.

Screenshot



Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"Dial"`
EZ_NAME	name	`"dial"`
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	
EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	
EZ_BORDER_WIDTH	borderWidth	`0`
EZ_BORDER_TYPE	borderType	`none`
EZ_FOCUS_PAD	focusPad	`2`
EZ_PADX	padx	`8`
EZ_PADY	pady	`6`
EZ_IPADX	iPadx	`0`
EZ_IPADY	iPady	`8`

EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`true`
EZ_TRANSIENT	transient	`false`

EZ_FONT_ID	fontID	`3`
EZ_FONT_NAME	fontName	`"*Bold*"`
EZ_LABEL_STRING	labelString	`null`

EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`

EZ_CALLBACK	N/A	`null null`
EZ_MOTION_CALLBACK	N/A	`null null`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`

EZ_FOREGROUND	foreground	`"black"`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`
EZ_DIAL_COLOR	dialColor	
EZ_INDICATOR_COLOR	indicatorColor	
EZ_SHAPED_WINDOW	shapedWindow	`false`
EZ_SHAPE_FILE	shapeFile	`NULL`
EZ_SHAPE_PIXMAP	N/A	`NULL`

EZ_BG_IMAGE_FILE_B	bgImageFileB	`null`
EZ_BG_PIXMAP_B	N/A	`none`
EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`

EZ_BACKING_STORE	backingStore	`true`

EZ_DIAL_RESOLUTION	dialResolution	`-1.0`
EZ_DIAL_RANGE	dialRange	`0.0 10.0`
EZ_DIAL_VALUE	dialValue	`0.0`
EZ_DIAL_DISPLAY_VALUE	dialDisplayValue	`10`
EZ_DIAL_STYLE	dialStyle	`0`
EZ_SHOW_TICKS	showTick	`10 10`

4.35 Gradient Bar

A gradient bar displays a smooth shaded color bar. The color bar is controled by up to 12 colors. A gradient bar behaves more or less like a slider, except it returns both a float point value and color value. The scroll button on a gradient bar has 4 different styles.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_GRADIENT_BAR`
Default instance name:	`"gradientBar"`
Default Class name:	`"GradientBar"`

Widget Interface Routines

`void EZ_SetGradientBarValue(EZ_Widget *widget, float value);`

set gradient bar value.

`float EZ_GetGradientBarValue(EZ_Widget *widget);`

retrieve the gradient bar value.

`float EZ_SetGradientBarRange(EZ_Widget *widget, float frm, float to);`

set gradient bar range.

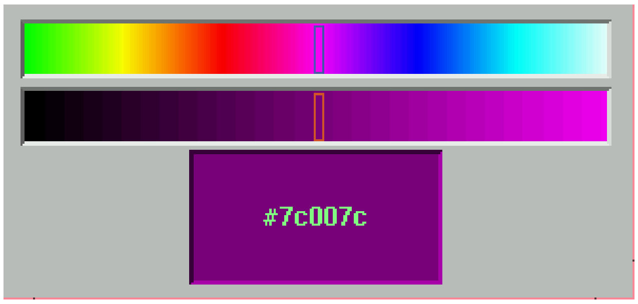
`void EZ_SetGradientBarColors(EZ_Widget *widget, char **colors, int ncolors);`

set gradient bar colors. `ncolors` is should be smaller than 12.

`void EZ_GetGradientBarColor(EZ_Widget *widget, int *r, int *g, int *b);`

return the selected color.

Screenshot



Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"GradientBar"`
EZ_NAME	name	`"gradientBar"`
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	
EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	

EZ_BORDER_WIDTH	borderWidth	`0`
EZ_BORDER_TYPE	borderType	`none`
EZ_FOCUS_PAD	focusPad	`2`

EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`true`
EZ_TRANSIENT	transient	`false`

EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`

EZ_CALLBACK	N/A	`null null`
EZ_MOTION_CALLBACK	N/A	`null null`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`

EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`
EZ_SHAPED_WINDOW	shapedWindow	`false`
EZ_SHAPE_FILE	shapeFile	`NULL`
EZ_SHAPE_PIXMAP	N/A	`NULL`

EZ_BG_IMAGE_FILE_B	bgImageFileB	`null`
EZ_BG_PIXMAP_B	N/A	`none`
EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`

EZ_BACKING_STORE	backingStore	`true`

EZ_SLIDER_STYLE	sliderStyle	`0`
EZ_SLIDER_INIT_VALUE	sliderInitValue	`0.5`

EZ_BAR_EDITABLE	barEditable	`True`
EZ_BAR_ORIENTATION	barOrientation	`EZ_HORIZONTAL`
EZ_BAR_WIDTH	barWidth	`4`
EZ_BAR_RANGE	barRange	`0.0 1.0`
EZ_BAR_COLOR_N	barColorN	`0,green`
EZ_BAR_COLOR_N	barColorN	`1,yellow`

4.36 Histogram

A histogram displays an array of vertical or horizontal bars, representing an array of floating point values. The size of the array, the number of bins is controled by the `EZ_HISTOGRAM_SAMPLE` resource. If this resource value is 0, it displays a dynamic array. The range of the histogram is controlled by the `EZ_BAR_RANGE` resource. The default range is [0, 100]. The width of the bars is controlled by the `EZ_BAR_WIDTH` resource. The colors of the bars is set by the `EZ_BAR_COLOR_N` resource.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_HISTOGRAM`
Default instance name:	`"histogram"`
Default Class name:	`"Histogram"`

Widget Interface Routines

`void EZ_SetHistogramValues(EZ_Widget *widget, float *values, int nvalues);`

set histogram bin values.

`void EZ_SetHistogramValue(EZ_Widget *widget, float value);`

set the value for the next bin. Used in progressing histograms.

`float EZ_GetHistogramValue(EZ_Widget *widget, int n);`

retrieve the nth value.

`int EZ_GetHistogramSelection(EZ_Widget *widget, float *value_return);`

retrieve the last selected bin and its value.

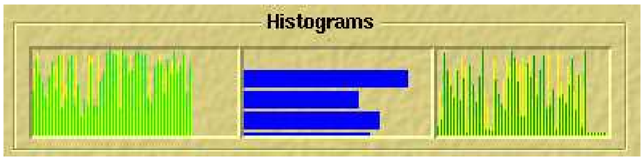
`float EZ_SetHistogramRange(EZ_Widget *widget, float frm, float to);`

set the range.

`void EZ_SetHistogramColors(EZ_Widget *widget, unsigned long *colors, int ncolors);`

set bar colors, `ncolors` is should be smaller than 12.

Screenshot



Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"GradientBar"`
EZ_NAME	name	`"gradientBar"`

EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	

EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	

EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	

EZ_BORDER_WIDTH	borderWidth	`0`
EZ_BORDER_TYPE	borderType	`sunken`

EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`true`
EZ_TRANSIENT	transient	`false`

EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`

EZ_CALLBACK	N/A	`null null`
EZ_MOTION_CALLBACK	N/A	`null null`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`

EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`
EZ_SHAPED_WINDOW	shapedWindow	`false`
EZ_SHAPE_FILE	shapeFile	`NULL`
EZ_SHAPE_PIXMAP	N/A	`NULL`

EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`

EZ_BACKING_STORE	backingStore	`true`

EZ_BAR_EDITABLE	barEditable	`True`
EZ_BAR_ORIENTATION	barOrientation	`EZ_HORIZONTAL`
EZ_BAR_WIDTH	barWidth	`1`
EZ_BAR_RANGE	barRange	`0.0 100.0`
EZ_BAR_COLOR_N	barColorN	`0,green`
EZ_BAR_COLOR_N	barColorN	`1,yellow`
EZ_HISTOGRAM_SAMPLE	histogramSample	`10`

4.37 Spread Sheet

The spread widget behaves like a two dimensional array of simplified input fields called cells. Cells can be left, right or center justified and can be displayed using different text properties. Rectanglar regions can be selected, sorted and charted. Contents of a spread sheet can be read from or exported to a text file (treated as a single string). Insertion/exporting is controled by the current tab and newline characters, the defaults are `\t` and `\n`.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_SPREAD_SHEET`
Default instance name:	`"spreadSheet"`
Default Class name:	`"SpreadSheet"`

Widget Interface Routines

```
void  EZ_SSheetGetSizes(EZ_Widget *ssheet, int *nrows, int *ncolumns);
```

return the current effect size of the spread sheet. The effect size is the minimal rectangle which contains all nonempty cells.

```
void  EZ_SSheetSetFirstCell(EZ_Widget *ssheet, int row, int col);
```

Set the the cell to be displayed at the upper left corner of the widget window.

```
void  EZ_SSheetSetTextProperty(EZ_Widget *ssheet, TextProperty *prop);
```

set the default text property. Newly created cells will be displayed using this default text property.

```
void  EZ_SSheetSetHScrollbar(EZ_Widget *ssheet, EZ_Widget *hscrollbar);
void  EZ_SSheetSetVScrollbar(EZ_Widget *ssheet, EZ_Widget *vscrollbar);
```

set the horizontal/vertical scrollbar for the spread sheet.

```
void  EZ_SSheetHideColumns(EZ_Widget *ssheet, int from, int to);
void  EZ_SSheetHideRows(EZ_Widget *ssheet, int from, int to);
void  EZ_SSheetUnhideColumns(EZ_Widget *ssheet, int from, int to);
void  EZ_SSheetUnhideRows(EZ_Widget *ssheet, int from, int to);
```

hide/unhide the specified columns/rows. Hidden columns/rows will be displyed using a vertical/horizontal line of width 1.

```
void  EZ_SSheetSwapRow(EZ_Widget *ssheet, int r1, int r2);
void  EZ_SSheetSwapColumn(EZ_Widget *ssheet, int c1, int c2);
void  EZ_SSheetSwapCell(EZ_Widget *ssheet, int r1, int c1, int r2, int c2);
```

Swap two rows/columns/cells.

```
void  EZ_SSheetClearRegion(EZ_Widget *ssheet, int r1, int c1, int r2, c2);
```

clear the region $[r1, c1] \times [r2, c2]$ inclusive.

```
void  EZ_SSheetCopyRegion(EZ_Widget *ssheet, int r1, int c1, int r2, int c2,
                          int r, int c, int delete);
```

copy the region $[r1, c1] \times [r2, c2]$ inclusive and paste it at $[r, c]$. If `delete` is true, clear the region $[r1, c1] \times [r2, c2]$ before pasting.

```
void  EZ_SSheetSortRegion(EZ_Widget *ssheet, int r1, int c1, int r2, int c2,
                        int keycolumn, int (*cmp)(CONST void *, CONST void *) );
```

Sort the specified region according to cells in `keycolumn`, using the supplied comparision function `cmp`. Cells are treated as strings in comparision. If `cmp` is NULL, it sorts the `keycolumn` in lexical ascending order.

```
void  EZ_SSheetSortRegionRow(EZ_Widget *ssheet, int r1, int c1, int r2, int c2,
                        int keyrow, int (*cmp)(CONST void *, CONST void *) );

void  EZ_SSheetSortColumns(EZ_Widget *ssheet, int r1, int c1, int r2, int c2,
                         int (*cmp)(CONST void *, CONST void *) );

void  EZ_SSheetSortRows(EZ_Widget *ssheet, int r1, int c1, int r2, int c2,
                        int (*cmp)(CONST void *, CONST void *) );

void  EZ_SSheetFindCell(EZ_Widget *ssheet, char *str, int *row, int *col);
```

find the first cell which contain `str` as a substring, and return the row and column index of the cell. This function returns 1 if a match is found and returns 0 otherwise.

```
void  EZ_SSheetSetTabChar(EZ_Widget *ssheet, char tab, char newline);
```

set the tab and line separator. They are used in `EZ_SSheetGetRegionString**` and `EZ_SSheetInsertRegionString**`. The default values are '
t' and '
n'.

```
char *EZ_SSheetGetRegionString(EZ_Widget *ssheet, int r1, int c1, int r2, int c2);
char *EZ_SSheetGetRegionStringWithProperty(EZ_Widget *ssheet, int r1, int c1,
              int r2, int c2, EZ_TextProperty ***prop);
```

return the specified region as a single string. Columns and rows are separated using the current tab and newline character. The second function returns the associated text property also.

```
char *EZ_SSheetInsertRegionString(EZ_Widget *ssheet, int r1, int c1, char *str);
char *EZ_SSheetInsertRegionStringWithProperty(EZ_Widget *ssheet, int r1, int c1,
              char *str, EZ_TextProperty **prop);
```

Insert cells represented by `str` at location [r1 c1]. `str` is parsed using the current tab and newline character. In the second version, inserted cells will be displayed using the associated text property.


```
char *EZ_SSheetInsertRegionString(EZ_Widget *ssheet, int r1, int c1, char *str);
char *EZ_SSheetInsertRegionStringWithProperty(EZ_Widget *ssheet, int r1, int c1,
              char *str, EZ_TextProperty **prop);
```

Insert cells represented by `str` at location [r1 c1]. `str` is parsed using the current tab and newline character. In the second version, inserted cells will be displayed using the associated text property.


```
int EZ_SSheetFindCursorCell(EZ_Widget *ssheet, int x, int y, int *row, int *col);
```

Find and return the indices of the cell that contains the point (x,y). This function returns 0 if no cells are found.

```
void EZ_SSheetHighlightRegion(EZ_Widget *ssheet, int r1, int c1, int r2, int c2);
```

Highlight the specified region. If r1 ¡ 0, clear the current highlighted region.

```
int EZ_SSheetGetHighlightedRegion(EZ_Widget *ssheet, int *r1, int *c1, int *r2, int *c2);
```

return the current highlight region. If no region is currently highlighted, returns 0.

```
void EZ_SSheetGetActiveCell(EZ_Widget *ssheet, int *row, int *col, char **str);
void EZ_SSheetGetActiveCellWithProperty(EZ_Widget *ssheet, int *row, int *col,
                                        char **str, EZ_TextProperty ***prop);
```

return the indices of the current active cell. If `str` is not null, it returns the active cell content.

```
void EZ_SSheetSetActiveCell(EZ_Widget *ssheet, int row, int col);
```

set the current active cell.

```
char *EZ_SSheetGetCell(EZ_Widget *ssheet, int row, int col);
char *EZ_SSheetGetCellWithProperty(EZ_Widget *ssheet, int row, int col, EZ_TextProperty *p);
```

return the content of the specified cell.

```
void EZ_SSheetDrawCell(EZ_Widget *ssheet, int row, int col, Drawable drawable,
                       int x, int y, int w, int h);
```

draw the content of the specified cell in drawable. (x,y,w,h) specifies the rectangle to render the text into.

```
void EZ_SSheetSetRegionJustification(EZ_Widget *ssheet, int r1, int c1,
                                    int r2, int c2, int justification);
```

justify the region. `justification` is one of `EZ_LEFT`,`EZ_RIGHT` or `EZ_CENTER`.

```
void EZ_SSheetSetRegionProperty(EZ_Widget *ssheet, int r1, int c1,
                                    int r2, int c2, EZ_TextProp *prop);
```

set and redisplay the specified region using the specified text property.

```
void EZ_SSheetMakeChart(EZ_Widget *ssheet, int r1, int c1, int r2, int c2,
            EZ_SchartHandle handle, int kind, void *cdata, int cmode);
   typedef  int  (*EZ_SChartHandle)(EZ_Widget *ss, void *chart, int r1, int c1,
                              int r2, int c2, int kind, void *data);
```

```
void EZ_SSheetDeleteChart(EZ_Widget *ssheet, EZ_Item *chart);
```

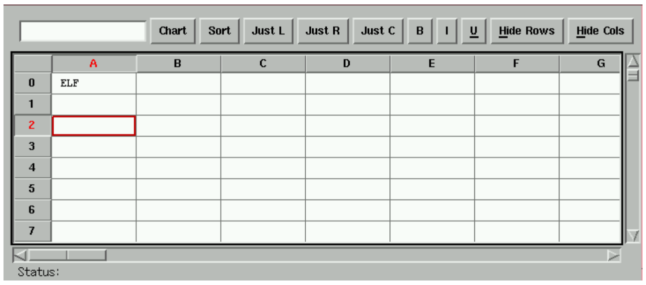
delete the specified chart.

`EZ_Item *EZ_SSheetGetActiveChart(EZ_Widget *ssheet);` return the currently selected chart.

`EZ_Item **EZ_SSheetGetAllCharts(EZ_Widget *ssheet, int *cnt);`
return the list of all charts and return the number of charts to `cnt`.

`void EZ_SSheetValidateCharts(EZ_Widget *ssheet, int r1, int c1, int r2, int c2);`
validate all charts whose definig region intersects to the specified region.

Screenshot



Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"SpreadSheet"`
EZ_NAME	name	`"spreadSheet"`
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	
EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	
EZ_BORDER_WIDTH	borderWidth	`0`
EZ_BORDER_TYPE	borderType	`none`
EZ_FOCUS_PAD	focusPad	`2`
EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`true`
EZ_TRANSIENT	transient	`false`
EZ_CURSOR	cursor	`"XC_C_PLUS"`
EZ_BUBBLE_STRING	bubbleString	`null`
EZ_CALLBACK	N/A	`null null`
EZ_MOTION_CALLBACK	N/A	`null null`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`
EZ_SHAPED_WINDOW	shapedWindow	`false`
EZ_SHAPE_FILE	shapeFile	`NULL`
EZ_SHAPE_PIXMAP	N/A	`NULL`

EZ_BG_IMAGE_FILE_B	bgImageFileB	`null`
EZ_BG_PIXMAP_B	N/A	`none`
EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`

EZ_BACKING_STORE	backingStore	`true`

EZ_SHEET_HEADER_SIZE	sheetHeaderSize	`32 16`
EZ_SHEET_CELL_SIZE	sheetCellSize	`96 24`

4.38 Thumb Wheel

A thumb wheel is a slider type widget that is used to edit a floating point value.

Widget Info

Symbolic Widget type:	`EZ_WIDGET_THUMB_WHEEL`
Default instance name:	`"thumbWheel"`
Default Class name:	`"ThumbWheel"`

Widget Interface Routines

`void EZ_SetThumbWheelValue(EZ_Widget *widget, float value);`

set the current thumb wheel value.

`float EZ_GetThumbWheelValue(EZ_Widget *widget);`

retrieve the current thumb wheel value.

`void EZ_SetThumbWheelRange(EZ_Widget *widget, float f, float t);`

set the range of a thum wheel.

Screenshot

Resources

Config_Option	*ResourceName*	*DefaultValue*
EZ_CLASS	class	`"ThumbWheel"`
EZ_NAME	name	`"thumbWheel"`
EZ_X	x	
EZ_Y	y	
EZ_WIDTH	width	
EZ_HEIGHT	height	
EZ_LOCATION	location	
EZ_SIZE	size	
EZ_GEOMETRY	geometry	
EZ_WIDTH_HINT	widthHint	
EZ_HEIGHT_HINT	heightHint	
EZ_SIZE_HINT	sizeHint	
EZ_BORDER_WIDTH	borderWidth	0
EZ_BORDER_TYPE	borderType	`none`
EZ_FOCUS_PAD	focusPad	2
EZ_PADX	padx	8
EZ_PADY	pady	6
EZ_IPADX	iPadx	0
EZ_IPADY	iPady	0
EZ_EXPAND	expand	`false`
EZ_PROPAGATE	propagate	`true`
EZ_TRANSIENT	transient	`false`

EZ_CURSOR	cursor	`"XC_left_ptr"`
EZ_BUBBLE_STRING	bubbleString	`null`

EZ_CALLBACK	N/A	`null null`
EZ_MOTION_CALLBACK	N/A	`null null`
EZ_DESTROY_CALLBACK	N/A	`null null`
EZ_EVENT_HANDLER	N/A	`null null`

EZ_FOREGROUND	foreground	`"black"`
EZ_BACKGROUND	background	`"gray74`
EZ_BG_IMAGE_FILE	bgImageFile	`null`
EZ_BG_PIXMAP	N/A	`none`
EZ_HIGHLIGHT_FOREGROUND	highlightForeground	
EZ_HIGHLIGHT_BACKGROUND	highlightBackground	

EZ_CLIENT_PTR_DATA	N/A	`null`
EZ_CLIENT_INT_DATA	clientIntData	`0`

EZ_DND_DRAG_CURSOR	dndDragCursor	`none`
EZ_DND_BUBBLE_STRING	dndBubbleString	`null`

EZ_BACKING_STORE	backingStore	`false`

EZ_SLIDER_RANGE	sliderRange	`0.0 1.0`
EZ_SLIDER_INIT_VALUE	sliderInitialValue	`0.0`

Chapter 5

General Interface Functions

This chapter lists and describes some general interface routines.

5.1 Getting the Version String

The next function returns the version string of *EZwgl*

```
char *EZ_LibraryVersion();
void EZ_GetVersionNumbers(int *major, int *minor);
```

5.2 Getting Info about the X Server

The display, screen, visual, visual depth etc. can be retrieved using the following functions.

```
void     EZ_GetDisplayInfo(Display **dpy, int *scrn,
                           Visual **vis, int *depth);

Colormap EZ_GetColormap();

void     EZ_GetDisplaySize(int *w, int *h);

Display *EZ_GetDisplay(); /* return the display */
Visual  *EZ_GetVisual();  /* return the visual  */
int      EZ_GetDepth();   /* return the v depth */
int      EZ_GetScreen();  /* return the scrn #  */
```

5.3 Allocating Colors

The following two routines can be used to allocate colors. They all return the pixel value for the specified color. If the exact color cannot be allocated, they return the closest match available.

```
unsigned long  EZ_AllocateColorFromRGB(int r, int g, int b);
unsigned long  EZ_AllocateColorFromName(char *name);
```

The next function converts a pixel value to RGB.

```
void  EZ_PixelValue2RGB(unsigned long pixel_value,
                        int *r, int *g, int *b);
```

5.4 Creating GC

```
GC   EZ_GetGC(unsigned long mask, XGCValues *gcv);
void EZ_FreeGC(GC gc);
```

`EZ_GetGC` finds or creates a GC with the specified attributes. One should not modify the returned GC.

The next few functions return GC related to the background of a widget.

```
int EZ_GetBackgroundGC(EZ_Widget *widget,
                  GC *gc, unsigned long *pv,
                  int highlight, int specialhighlight);
int EZ_GetBackgroundTileGC(EZ_Widget *widget,
                  GC *gc, unsigned long *pv,
                  int highlight, int specialhighlight);

int EZ_GetParentBgGC(EZ_Widget *widget,
                     GC *gc, unsigned long *pv);
int EZ_GetParentBgTileGC(EZ_Widget *widget,
                         GC *gc, unsigned long *pv);
```

The next function return the foreground color of a widget.

```
unsigned long EZ_GetForegroundPV(EZ_Widget *widget);
```

5.5 Setting and Extracting Client Data

The `EZ_Widget` data structure provides two special fields for applications to hook in application data to a widget. One is of type `int` and the other is of type `void *`. To set these fields, use

```
void   EZ_SetWidgetIntData(EZ_Widget *widget, int  i);
void   EZ_SetWidgetPtrData(EZ_Widget *widget, void *ptr);
void   EZ_SetWidgetClientData(EZ_Widget *widget, int i, void *ptr);
```

To extract client data from a widget, use

```
int    EZ_GetWidgetIntData(EZ_Widget *widget);
void  *EZ_GetWidgetPtrData(EZ_Widget *widget);
void   EZ_GetWidgetClientData(EZ_Widget *widget, int *i, void **ptr);
```

$\mathcal{EZ}$ widgets are also capable of allocating and managing dynamic storage. This feature provides a convenient way for an application to attach more complex client data to a widget without worrying about memory management. It is particularly convenient for situations when one needs to attach more than one pointers to a widget and doesn't want to introduce a new data structure for it.

The `EZ_Widget` data structure contains a pointer to `EZ_UnknownDataType` which serves as the hook for dynamically allocated storage.

```
typedef  union {
  char           c;
  short          s;
  int            i;
  long           l;
  unsigned char   uc;
  unsigned short  us;
  unsigned int    ui;
  unsigned long   ul;
  float           f;
  void           *p;
}  EZ_UnknownDataType;
```

Here are the relevent interface functions.

```
void EZ_SetWidgetUnknownData(EZ_Widget *widget, int idx,
                             EZ_UnknownDataType value);
```

(re)allocate (if needed) and set the the `idx`th element to `value`.

```
int EZ_GetWidgetUnknownData(EZ_Widget *widget, int idx,
                            EZ_UnknownDataType  *ret);
```

retrieve the `idx`th element. Return 1 if succesful, return 0 otherwise.

```
EZ_UnknownDataType   *EZ_GetWidgetUnknownDataPtr(EZ_Widget *widget);
```

return the dynamically allocated array.

5.6 Inquring the Geometry of a Widget

The *EZ* widget library provides a few routines for extracting the geometry of a widget.

```
void EZ_GetWidgetGeometry(EZ_Widget *widget, int *x_return,
            int *y_return, int *w_return, int *h_return);
void EZ_GetWidgetAbsoluteGeometry(EZ_Widget *widget, int *x_return,
            int *y_return, int *w_return, int *h_return);
```

return the geometry of a widget.

```
void EZ_GetWidgetPosition(EZ_Widget *widget, int *x, int *y);
```

return the location of (the upper left corner) a widget.

```
void EZ_GetWidgetDimension(EZ_Widget *widget, int *w, int *h);
void EZ_GetWidgetMinDimension(EZ_Widget *widget, int *w, int *h);
```

return the dimension (or minimal dimension) of a widget.

```
int  EZ_GetWidgetBorderWidth(EZ_Widget *widget);
void EZ_GetWidgetBorder(EZ_Widget *widget,int *bwidth,
                                          int *btype);
```

return the border characteristics.

```
void  EZ_GetWidgetPaddings(EZ_Widget *widget, int *padx,
                             int *pady, int *focus_pad);
```

return all paddings.

5.7 Get Parent/Children Widgets

```
EZ_Widget *EZ_GetParentWidget(EZ_Widget *widget);
EZ_Widget *EZ_GetChildrenWidget(EZ_Widget *widget);
EZ_Widget *EZ_GetSiblingWidget(EZ_Widget *widget);
```

5.8 Disable/Enable a Widget

EZwgl provides two pairs of routines to control the state of *EZ* widgets. A widget can be disabled or de-activated. A disabled widget will not handle user inputs. A de-activated widget will no longer be displayed until re-activated and re-displayed.

```
void EZ_DisableWidget(EZ_Widget *widget);
void EZ_EnableWidget(EZ_Widget *widget);

void EZ_DisableWidgetTree(EZ_Widget *widget);
void EZ_EnableWidgetTree(EZ_Widget *widget);


void EZ_DeActivateWidget(EZ_Widget *widget);
void EZ_ActivateWidget(EZ_Widget *widget);

void EZ_DeActivateChildren(EZ_Widget *widget);
void EZ_ActivateChildren(EZ_Widget *widget);

void EZ_DeActivateSibling(EZ_Widget *widget);
void EZ_ActivateSibling(EZ_Widget *widget);
```

5.9 Temporarily Freeze a Widget

At times, it is desirable to block a widget from being redisplayed for a short period of time. For example, during the process of inserting 100 items into a *listbox*, one usually don't want the list box be refreshed 100 times, but rather, be updated only once after the last item has been inserted. The *EZ* widget library provides a pair of routines to temporarily block a widget from being refreshed.

```
void EZ_FreezeWidget(EZ_Widget *widget);
void EZ_UnFreezeWidget(EZ_Widget *widget);
```

```
void EZ_FreezeDisplay(void);
void EZ_ThawDisplay(void);
```

5.10 Hide a Widget

For transient widgets such as dialogue boxes, one usually wants to remove them from the screen after interaction has completed. The *EZ* widget library provides one routine that does a non-destructive removal of a mapped widget.

```
void EZ_HideWidget(EZ_Widget *widget);
```

5.11 Destroy a Widget

To destroy a widget, use

```
void EZ_DestroyWidget(EZ_Widget *widget);
```

This function destroys a widget. All data allocated for widget is freed after this function returns. During the destruction process, it calls all the destroy callbacks registered to widget.

A destroyed widget should never be referenced again.

5.12 Grab a Widget

For dialogue boxes, one usually wants to suspend the application until the user responds to the dialogue. In particular, one needs to suspend event processing on all widgets except the dialogue. The latter can be achieved by using `EZ_SetGrab`.

```
void  EZ_SetGrab(EZ_Widget *widget);
void  EZ_ReleaseGrab(void);
```

`EZ_SetGrab` sets a grab on `widget`. It informs the *EZ* event dispatcher to drop all events not associated with `widget` or its descendants. The grab, if set succesfully, will remain in effect until either

- the widget becomes unviewable. I.e., the widget window or one of its ancestor's window is unmapped; or
- a call to `EZ_ReleaseGrab`.

5.13 Set Keyboard Focuse to a Widget

The next function set the keyboard focus to the specified widget if the widget is focusable.

```
void       EZ_SetFocusTo(EZ_Widget *widget);
EZ_Widget *EZ_GetFocus(void);
```

5.14 Communication Window

Each *EZwgl* has an unmapped window for communication (e.g Drag and Drop). The next function returns the window ID of this window.

```
Window  EZ_GetCommunicationWindow();
```

5.15 Cursor Functions

```
Cursor EZ_GetCursor(int type); /* see <X11/cursorfont.h> */
Cursor EZ_GetCursorByName(char *name);
char  *EZ_GetCursorName(Cursor csr);

Cursor EZ_CreateCursorFromBitmapFile(char *file, char *maskFile,
                                      char *fg, char *bg,
                                      int  hot_x, int hot_y,
                                      char *name, int *idx);
Cursor EZ_CreateCursorFromBitmapData(int wid, int hig,
                                     char *data, char *maskData,
                                     char *fg, char *bg,
                                     int  hot_x,   int hot_y,
                                     char *name,int *idx_ret);
```

5.16 Grab Pointer

```
int   EZ_GrabPointer(EZ_Widget *widget, Cursor csr, int timeout);
void  EZ_UngrabPointer(void);
void  EZ_BusyCursor(EZ_Widget *widget, Cursor cursor, int timeout);
void  EZ_WaitCursor(EZ_Widget *widget, Cursor cursor);
void  EZ_NormalCursor (EZ_Widget *widget);
```

5.17 Warp Pointer

```
void EZ_WarpPointer(int x, int y);
void EZ_WarpPointerToWidgetWindow(EZ_Widget *wgt, int where);
```

5.18 The Bubble Help Widget

The bubble help widget is a toplevel FreeLabel.

```
EZ_Widget *EZ_GetBubbleHelp(void);
```

5.19 Display A Widget

```
void  EZ_DisplayWidget(EZ_Widget *widget);
void  EZ_DisplayWidgetUnderPointer(EZ_Widget *widget, int xoff, int yoff);

void  EZ_BlinkWidget(EZ_Widget *widget, int cnt);
void  EZ_FlashWidgetBorder(EZ_Widget *widget);
void  EZ_HighlightWidget(EZ_Widget *widget, int on_off);

int   EZ_WidgetIsViewable(EZ_Widget *widget);

void  EZ_RaiseWidgetWindow(EZ_Widget *widget);
void  EZ_RedisplayWidgetTree(EZ_Widget *widget);
```

5.20 Overriding the Default Attributes

There are a few global attributes for the $\mathcal{EZ}$ widget library. They affect the appearance and the behavior of $\mathcal{EZ}$ widgets. Some of these attributes can be modified at the initialization time.
Note: all the global attributes can be altered via resources. Procedures listed in this section take procedence over resources.

It is recommended you use resource files to override these default attributes instead of hardcoding them by calling routines described in this section.

5.20.1 Setting the Default Background

The *EZ* Widget library provides a function to set the **default** background color for all widgets in an application.

```
int EZ_SetGlobalBackground(char *colorName);
```

The resource that sets the default background color

```
application.background: color
```

This function alters the five color entires allocated for widget background and borders. The RGB intensities for `colorName` should be in the middle range of the interval [0,255].

This function has no effect if the `-bg` command line option is properly set. Neither does it have effects on widgets whose background or their ancestors' background are hard coded.

5.20.2 Setting the Default Fonts

EZwgl uses three default fonts to display text.

```
"-Adobe-Helvetica-Bold-R-Normal--*-120-*-*-*-*-*-*",
"-*-Courier-Medium-R-Normal--*-120-*-*-*-*-*-*"
"-Adobe-Times-Bold-I-Normal--*-120-*-*-*-*-*-*",
```

The first one (bold) is used for displaying textual labels; the second one (fixed width) is used for displaying text in text widgets and input fields and the last one (bold italic) font is used for displaying menu titles. One can override these default fonts by the following set of functions. All these functions overrides the corresponding resources set by the application default file.

```
void  EZ_SetDefaultLabelFont(char *fontName);
void  EZ_SetDefaultEntryTextFont(char *fontName);
void  EZ_SetDefaultMenuTitleFont(char *fontName);
```

The resources that set these default fonts are:

```
application.labelFont:     font_name
application.textFont:      font_name
application.menuTitleFont: font_name
```

5.20.3 Disable Button Highlighting/Offseting

By default, a button type widget lights up when the pointer is over it; and its label shifts a little bit to the right when it is pressed. You can control these behavior via the following two pair of functions.

```
void EZ_DisableHighlight(void);
void EZ_EnableHighlight(void);
void EZ_DisableLabelOffset(void);
void EZ_EnableLabelOffset(void);
```

The relevent resources are

```
application.highlight:     boolean
application.labelOffset:   boolean
```

5.20.4 Disable Slider Depression

The default behavior of a slider/scrollbar is that when the slider button is pressed, it depresses, gives you the illusion of being pressed, just like a button. You can control the behavior by the following two functions.

```
void EZ_DisableSliderDepression(void);
void EZ_EnableSliderDepression(void);
```

The corresponding resource is

```
application.sliderDepression:     boolean
```

5.20.5 Set Keyboard Focus Highlight Color

The keyboard focus highlight color is controled by

```
void EZ_SetKbdHighlightColor(char *color_name);
```

The equivalent resource is:

```
application.focusHighlightColor:     color_name
```

5.20.6 Miscellaneous

The next three functions may be used to slide open/close a toplevel widget. The toplevel widget should have the `EZ_TRANSIENT` or `EZ_OVERRIDE_REDIRECT` resource set in order for them to work properly. Parameters are `x,y` the initial origin of the widget window, relative to the origin of the root window; `direction` the direction to slide open/close the widget window, values are one of `EZ_NORTH`, `EZ_NORTHEAST`, `EZ_EAST`, `EZ_SOUTHEAST`, `EZ_SOUTH`, `EZ_SOUTHWEST`, `EZ_WEST`, `EZ_NORTHWEST` or `-1`; `speed` is the number of pixels to slide over one step.

```
void EZ_SlideOpenWidget(EZ_Widget *w, int x, int y,
                        int direction, int speed);


void EZ_SlideOpenWidgetRelativeTo(EZ_Widget *widget, EZ_Widget *rel,
                                  int direction, int speed);


void EZ_SlideCloseWidget(EZ_Widget *widget, int direction, int speed);
```

Chapter 6

Write Your Own Widgets

This chapter explains the mechanism provided by *EZwgl* for implementing your own widgets.

6.1 Write Primitive Widgets

To write your own primitive widget, you need to

- select a default class name and instance name for your widget.
- provide the following 5 procedures

```
          setup(EZ_Widget *widget);
    computeSize(EZ_Widget *widget,  int *w, int *h);
       freeData(EZ_Widget *widget);
           draw(EZ_Widget *widget);
    eventHandle(EZ_Widget *widget, XEvent *event)
```

  `setup` is invoked whenever an instance of your widget is created. You should include here the widget or widget class initialization code and the code for parsing private resources. Primitive widgets are implemented internally through the `RawXWindow` widget. The widget data structure provides an array of 10 `EZ_UnknownDataType` for you to hook up private data to your widget.

  `computeSize` is invoked by the geometry manager to figure out the minimal size of your widget.

  `freeData` is invoked by `EZ_DestroyWidget` to clean up widget private data.

  `draw` is invoked whenever there are needs to repaint the widget window.

  `evehtHandle` is the default widget event handler.

- call `EZ_RegisterPrimitiveWidget` to register your widget. This function returns an integer identifier.
- write interface routines as needed.

After a widget has been registered, it can be instantiatd by calling `EZ_CreateWidgetXrm` with type the returned identifier.

6.1.1 Example: Locator

In this section, we implement the locator widget, a widget selects the mouse pointer coordinates in a rectangular area, scaled to [0,1]x[0,1].

Our locator has one private resources, `value`, which takes a pair of float point values in the range between 0.0 and 1.0.

We implement two interface routines for our locator. `ez_GetLocatorXY` and `ez_SetLocatorXY`. The first one returns the coordinates of the locator and the second one sets the coordinates of the locator.

Our locator will handle ButtonPress and KeyPress events. It also support MotionCallbacks, private event handlers and drag and drop.

```
/************************ Locator Widget ***************************/
#include "EZ.h"

void ez_GetLocatorXY (EZ_Widget *widget, float *x, float *y);
void ez_SetLocatorXY (EZ_Widget *widget, float x,  float y);
/*******************************************************************/
```

```
int  Locator_TYPE;

/*******************************************************************/
static void LocatorSetup       (EZ_Widget *widget);
static void LocatorFreeData    (EZ_Widget *widget);
static void LocatorComputeSize (EZ_Widget *widget, int *w, int *h);
static void LocatorDraw        (EZ_Widget *widget);
static void LocatorEventHandle (EZ_Widget *widget, XEvent *event);
/*******************************************************************/

static void LocatorSetup(EZ_Widget *widget)
{
  /* this procedure is invoked when a new LOCATOR widget
   * is created. We initialize the locator through
   * resources and create a gc for rendering the crosshair.
   *
   * primitive widgets are implemented using the RAW_XWINDOW
   * widget. The RAW_XWINDOW widget data structure has
   * an array of 10 EZ_UnknownDataType reserved for
   * us to hook up private data.
   */
  EZ_UnknownDataType  *data = EZ_GetRawXWinPrivateData(widget);
  unsigned long       mask  = 0L;
  XGCValues           gcvalues;
  char                *value;   /* tmp return for resource */

  /* data is an array of 10 EZ_UnknownDataTypes */
  data[0].f = 0.5;
  data[1].f = 0.5;
  /* parse private resources */
  if(EZ_GetWidgetResource(widget, "value", &value))
    {
      float   x,y;
      if(sscanf(value, "%f %f", &x, &y) == 2)
        {
          if( x >= 0.0 && x <= 1.0) data[0].f = x;
          if( y >= 0.0 && y <= 1.0) data[1].f = y;
        }
    }
  data[2].p = (void *)EZ_GetGC(mask, &gcvalues);
}
/***************************************************************/

static void LocatorFreeData(EZ_Widget *widget)
{
  /* this procedure is called when a LOCATOR widget is
   * destroyed. We free the GC allocated for this widget
   */
  EZ_UnknownDataType  *data = EZ_GetRawXWinPrivateData(widget);
  GC gc = (GC)data[2].p;
  /* for gc created by EZ_GetGC, you must use EZ_FreeGC to free it ! */
  EZ_FreeGC(gc);
}
/***************************************************************/

static void LocatorComputeSize(EZ_Widget *widget, int *w, int *h)
{
  /* this procedure is invoked by the geometry manager to
   * figure out the minimal size of LOCATOR.
   */
  *w =100; *h =100;
}
/***************************************************************/

static void LocatorDraw(EZ_Widget *widget)
{
  int focus_pad = 0;   /* kbd focus highlight padding */
  int border    = 0;   /* border style */
  int bwidth    = 0;   /* border width */
  int padx, pady;      /* XY padding, not used */
  int width, height;   /* widget window width and height */
  Display  *dpy;       /* the display */
  Window   win;        /* widget window */
  int      depth;      /* depth of visual */
  int      tile;       /* is bg tiled ? */
  int      x, y, tmp, tmp2;
  Pixmap   pixmap;
  GC       gc;
  unsigned long pv;
  EZ_UnknownDataType  *data = EZ_GetRawXWinPrivateData(widget);

  EZ_GetWidgetPaddings(widget, &padx, &pady, &focus_pad); /* get all paddings       */
  EZ_GetWidgetBorder(widget, &bwidth, &border);           /* border characteristics */
  EZ_GetWidgetDimension(widget, &width, &height);         /* widget size            */
  dpy = EZ_GetDisplay();              /* display   */
  depth = EZ_GetDepth();             /*   visual    */
  win = EZ_GetWidgetWindow(widget); /* widget window */
  tmp = focus_pad + bwidth;
  tmp2 = tmp + tmp;
```

```
  /* draw into a tmp pixmap, copy to the widget window when done */
  pixmap = XCreatePixmap(dpy, win, width, height, depth);

  /* fill the background */
  EZ_GetBackgroundTileGC(widget, &gc, &pv, 0,0);
  XFillRectangle(dpy, pixmap, gc, 0, 0, width, height);

  /* fix kbd focus highlight border */
  EZ_FixFocusHighlightBorder(widget, pixmap, width, height, focus_pad);

  /* render the foreground, the crosshair */
  gc = (GC)data[2].p;
  pv = EZ_GetForegroundPV(widget);
  XSetForeground(dpy, gc, pv);

  x = tmp + (int)((width - tmp2) * data[0].f + 0.5);
  y = tmp + (int)((height- tmp2) * data[1].f + 0.5);

  XDrawLine(dpy, pixmap, gc, x, tmp, x, height -tmp);
  XDrawLine(dpy, pixmap, gc, tmp, y, width - tmp,y);

  /* render the widget border */
  EZ_DrawRectBorder(widget, pixmap);

  /* and finally  ... */
  XCopyArea(dpy,pixmap,win, gc,0,0,width,height,0,0);
  XFreePixmap(dpy, pixmap);
}
/****************************************************************/

static void LocatorEventHandle(EZ_Widget *widget, XEvent *event)
{
  if(event->type == Expose) LocatorDraw(widget);
  else if(event->type == ButtonPress || event->type == KeyPress)
    {
      int padx, pady, focus_pad=0;               /* paddings  */
      int border = 0, bwidth = 0, bwpad, bwpad2;  /* border .. */
      int x,y, width, height;                     /* dimension */
      Display *dpy = EZ_GetDisplay();             /* display   */
      XEvent xevent;
      float save_x, save_y;
      EZ_UnknownDataType  *data = EZ_GetRawXWinPrivateData(widget);

      if(event->type == ButtonPress)
        {
          if(event->xbutton.button == EZ_Btn1)
            {
              /* get paddings and  dimension */
              EZ_GetWidgetPaddings(widget, &padx, &pady, &focus_pad);
              EZ_GetWidgetBorder(widget, &bwidth, &border);
              EZ_GetWidgetDimension(widget, &width, &height);
              bwpad = focus_pad + bwidth;
              bwpad2 = bwpad + bwpad;
              x = event->xbutton.x;
              y = event->xbutton.y;
              save_x = data[0].f;
              save_y = data[1].f;
              while(1)
                {
                  /* loop and wait for ButtonRelease. Invoke
                   * private event handlers on other events. Handle
                   * MotionNotify events and execute MotionCallbacks
                   * if needed.
                   */
                  if(x >= 0)
                    {
                      data[0].f = (float)(x-bwpad)/(float)(width  - bwpad2);
                      data[1].f = (float)(y-bwpad)/(float)(height - bwpad2);
                      if(data[0].f < 0.0) data[0].f = 0.0; else if(data[0].f > 1.0) data[0].f=1.0;
                      if(data[1].f < 0.0) data[1].f = 0.0; else if(data[1].f > 1.0) data[1].f=1.0;
                      LocatorDraw(widget);
                      /* if locator has changed it state, invoke motion callbacks */
                      if(save_x != data[0].f || save_y != data[1].f)
                        EZ_CallWidgetMotionCallbacks(widget);
                    }
                  XNextEvent(dpy, &xevent);

                  /* tricky! We want to give private EHandlers a chance  */
                  if(EZ_FilterEvent(&xevent)) EZ_InvokePrivateEventHandler(&xevent);
                  if(xevent.type == Expose)
                    {
                      EZ_WidgetDispatchEvent(&xevent);
                      EZ_RemoveEvent(&xevent);
                    }
                  if(xevent.type == MotionNotify)
                    {
                      x = xevent.xmotion.x;
                      y = xevent.xmotion.y;
                      while(XCheckTypedEvent(dpy, MotionNotify, &xevent));
                    }
```

```
                else if(xevent.type == ButtonRelease && xevent.xbutton.button == Button1)
                  break;
                else x = -1111;
              }
          }
        else if(event->xbutton.button == EZ_Btn3)
          {
            EZ_HandleDnDEvents(widget, event);
          }
      }
    else /* key press */
      {
        char              tmpbuffer[8];
        int               buffersize = 8, xmove_unit, ymove_unit, count;
        KeySym            keysym;
        XComposeStatus    compose;

        count = XLookupString(&(event->xkey), tmpbuffer, buffersize, &keysym, &compose);
        tmpbuffer[count] = '\0';
        xmove_unit = 0;
        ymove_unit = 0;

        switch(keysym)
          {
          case XK_Left: case XK_b:  case XK_h: case XK_B:  case XK_H:
#ifdef XK_KP_Left
          case XK_KP_Left:
#endif
            if(event->xkey.state & ShiftMask)        xmove_unit = -4;
            else if(event->xkey.state & Mod1Mask)    xmove_unit = -10 ;
            else if(event->xkey.state & ControlMask) xmove_unit = -20;
            else xmove_unit = -1;
            break;
          case XK_Right: case XK_f: case XK_l: case XK_F: case XK_L:
#ifdef XK_KP_Right
          case XK_KP_Right:
#endif
            if(event->xkey.state & ShiftMask)         xmove_unit = 4;
            else if(event->xkey.state & Mod1Mask)     xmove_unit = 10 ;
            else if(event->xkey.state & ControlMask)  xmove_unit = 20;
            else xmove_unit = 1;
            break;
          case XK_Up: case XK_k:  case XK_p: case XK_K:  case XK_P:
#ifdef XK_KP_Up
          case XK_KP_Up:
#endif
            if(event->xkey.state & ShiftMask)         ymove_unit = -4;
            else if(event->xkey.state & Mod1Mask)     ymove_unit = -10 ;
            else if(event->xkey.state & ControlMask)  ymove_unit = -20;
            else ymove_unit = -1;
            break;
          case XK_Down: case XK_n: case XK_j: case XK_N: case XK_J:
#ifdef XK_KP_Down
          case XK_KP_Down:
#endif
            if(event->xkey.state & ShiftMask)         ymove_unit = 4;
            else if(event->xkey.state & Mod1Mask)     ymove_unit = 10 ;
            else if(event->xkey.state & ControlMask)  ymove_unit = 20;
            else ymove_unit = 1;
            break;
          default:
            break;
          }

        if(xmove_unit | ymove_unit)
          {
            data[0].f += (xmove_unit) * 0.01;
            data[1].f += (ymove_unit) * 0.01;
            if(data[0].f < 0.0) data[0].f = 0.0; else if(data[0].f > 1.0) data[0].f=1.0;
            if(data[1].f < 0.0) data[1].f = 0.0; else if(data[1].f > 1.0) data[1].f=1.0;
            LocatorDraw(widget);
            EZ_CallWidgetMotionCallbacks(widget);
          }
        /* normally there are too many KeyPress events ... */
        while(XCheckTypedEvent(dpy, KeyPress, &xevent));
      }
  }
}
/*********************************************************************/

void ez_GetLocatorXY(EZ_Widget *widget, float *x, float *y)
{
  /* return the coordinates of the locator */
  EZ_UnknownDataType  *data = EZ_GetRawXWinPrivateData(widget);
  *x = data[0].f;  *y = data[1].f;
}
/*********************************************************************/

void ez_SetLocatorXY(EZ_Widget *widget, float x, float y)
{
```

```
  /* set the location of the locator */
  EZ_UnknownDataType  *data = EZ_GetRawXWinPrivateData(widget);
  if(x < 0.0) x = 0.0; else if(x > 1.0) x = 1.0;
  if(y < 0.0) y = 0.0; else if(y > 1.0) y = 1.0;
  if(x != data[0].f || y != data[1].f)
    {
      data[0].f = x;
      data[1].f = y;
      if(EZ_WidgetIsViewable(widget)) EZ_DrawWidget(widget);
      EZ_CallWidgetMotionCallbacks(widget);
    }
}
/*******************************************************************/
/* TEST */
main(int ac, char **av)
{
  EZ_Widget *locator;
  EZ_Initialize(ac, av, 0);
  LocatorType = EZ_RegisterPrimitiveWidget("locator", "Locator",
                             LocatorSetup, LocatorComputeSize,
            LocatorDraw, LocatorFreeData, LocatorEventHandle);
  locator = EZ_CreateWidgetXrm(LocatorType, NULL,
                              "aLocator", "ALocator",
                              0);
  EZ_DisplayWidget(locator);
  EZ_EventMainLoop();
}
/*******************************************************************/
```

6.1.2 Help Functions

In this section, we list the functions that are useful for writing primitive widgets.

`void EZ_GetWidgetPaddings(EZ_Widget *widget, int *padx, int *pady, int *focusPd);`

return the paddings of a widget.

`void EZ_GetWidgetBorder(EZ_Widget *widget, int *bwidth, int *btype));`

return the border characteristics.

`void EZ_GetWidgetDimension(EZ_Widget *widget, int *w, int *h);`

return the dimension of a widget.

`void EZ_GetWidgetMinDimension(EZ_Widget *widget, int *w, int *h);`

return the minimal dimension of a widget.

`void EZ_GetWidgetPosition(EZ_Widget *widget, int *x, int *y);`

return the location of a widget.

`void EZ_GetWidgetGeometry(EZ_Widget *widget, int *x,int *y, int *w, int *h);`

return the geometry of a widget.

```
int EZ_GetBackgroundTileGC(EZ_Widget *widget, GC *gc, unsigned long *pv,
                           int highlight, int force_highlight);
```

get background GC, return 1 if tile is used (with tile origin properly set).

```
int EZ_GetBackgroundGC(EZ_Widget *widget, GC *gc, unsigned long *pv,
                       int highlight, int force_highlight);
```

get background GC, return 1 if tile is used (with tile origin unspecified).

```
int EZ_GetDarkBrightNormalBDGC(EZ_Widget *widget, GC *dark,
                               GC *bright, GC *normal);
```

get border GCs, return 1 if tile is used (with tile origin unspecified).

`void EZ_GetWidgetTileOrigin(EZ_Widget *widget, int *x, int *y);`

if bg tile is used, return the tile origin.

```
int EZ_GetWidgetOrientation(EZ_Widget *widget);
```

return the orientation of a widget.

```
Window EZ_GetWidgetWindow(EZ_Widget *widget);
```

return the widget window.

```
EZ_Widget *EZ_GetParentWidget(EZ_Widget *widget);
```

return the parent widget.

```
EZ_Widget *EZ_GetChildrenWidget(EZ_Widget *widget);
```

return the first child, other children are siblings of the first child.

```
EZ_Widget *EZ_GetSiblingWidget(EZ_Widget *widget);
```

return the sibling widget.

```
unsigned long EZ_GetForegroundPV(EZ_Widget *widget);
```

return the pixel value of the foreground color.

```
EZ_Bitmap *EZ_GetWidgetPixmap(EZ_Widget *widget);
```

return the label pixmap of a widget.

```
void EZ_GetBrightBDpv(EZ_Widget *widget, unsigned long *pv);
```

return the pixel value of the bright side of the border.

```
void EZ_GetDarkBDpv(EZ_Widget *widget, unsigned long *pv);
```

return the pixel value of the dark side of the border.

```
XFontStruct *EZ_GetWidgetFont(EZ_Widget *widget);
```

return widget font.

```
Display *EZ_GetDisplay(void);
```

return the display.

```
void   EZ_GetDisplayInfo(Display **dpy, int *scrn, Visual **vis, int *depth);
```

return info about current display.

```
void  EZ_GetDisplaySize(int *w, int *h);
```

return the display screen size.

```
GC  EZ_GetGC(unsigned long mask, GCValues gcvalues);
```

create and return a GC.

```
void  EZ_FreeGC(GC gc);
```

free a GC created by `EZ_GetGC`.

```
int  EZ_RegisterPrimitiveWidget(char *iname, char *cname,
                void (*Setup)(EZ_Widget *widget),
                void (*ComputeSize)(EZ_Widget *widget, int *w, int *h),
                void (*DrawWidget)(EZ_Widget *widget),
                void (*FreeData)(EZ_Widget *widget),
               void  (*EventHandle)(EZ_Widget *widget, XEvent *event));
```

register a primitive widget to *EZwgl*. This function returns an integer widget type identifier. This type identifier can be used by `EZ_CreateWidget` to instantiate the registered widget.

6.2 Write Composite Widgets

To make your composite widget available to `EZ_CreateWidget`, you need to

- select a default class name and a default instance name for your widget.
- Provide a creation function, of the following type

  `EZ_Widget *create(EZ_Widget *parent, char *inst_name, char *class_name);`

- call `EZ_RegisterCompositeWidget` to register your widget.
- write interface routines as needed.

After a widget has been registered, it can be instantiatd by calling `EZ_CreateWidgetXrm` with the registered type.

6.2.1 Example: Counter

In this section, we implement a counter widget. Our counter has three private resources, `value` (an integer), `increment` (a pair of integers) and `counterWidth` (an integer specifies the width of the counter label).

We implement two interface procedures; `ez_GetCounter` and `ez_SetCounter`.

```
/************************ Counter Widget ******************************/
#include "EZ.h"

int  ez_GetCounter (EZ_Widget *widget);
void ez_SetCounter (EZ_Widget *widget, int value);
/*********************************************************************/

int Counter_TYPE;

/*********************************************************************/
static EZ_Widget *CounterCreate(EZ_Widget *parent, char *iname, char *cname);
/*********************************************************************/
typedef struct counter_ {
  EZ_Widget   *frame,  *label;
  EZ_Widget   *btnLL, *btnL, *btnR, *btnRR;
  int         type, value;
  int         increment1, increment2;
} ez_counter;
/*********************************************************************/

int  ez_GetCounter(EZ_Widget *widget)
{
  /* return counter value */
  ez_counter *ptr = (ez_counter *)EZ_GetWidgetPtrData(widget);
  return(ptr->value);
}
/*********************************************************************/

void  ez_SetCounter(EZ_Widget *widget, int value)
{
  /* set counter value and update display */
  ez_counter *ptr = (ez_counter *)EZ_GetWidgetPtrData(widget);
  if(ptr->value != value)
    {
      char str[64];
      ptr->value = value;
      sprintf(str, "%d", value);
      EZ_ConfigureWidget(ptr->label, EZ_LABEL_STRING, str, 0);
      if(EZ_WidgetIsViewable(ptr->label)) EZ_DrawWidget(ptr->label);
    }
}
/*********************************************************************/

static void btnEHandle(EZ_Widget *btn, void *data, int etype, XEvent *xev)
{
  /* the arrow buttons are labels, this event handler replaces
   * the default event handler
   */
  switch(xev->type)
    {
    case Expose:      EZ_DrawWidget(btn); break;
    case EnterNotify: EZ_HighlightWidget(btn, 1); break;
    case LeaveNotify: EZ_HighlightWidget(btn, 0); break;
    case ButtonPress:
      {
        if(xev->xbutton.button == EZ_Btn1)
          {
            Display    *dpy = EZ_GetDisplay();
            XEvent     xevent;
            ez_counter *ptr = (ez_counter *)data;
```

```
            int        whichBtn = EZ_GetWidgetIntData(btn);
            int        nnn = 10;
            int        increment, done = 0;

            switch(whichBtn)
              {
              case -2: increment = - ptr->increment2; break;
              case -1: increment = - ptr->increment1; break;
              case  1: increment =   ptr->increment1; break;
              case  2: increment =   ptr->increment2; break;
              default: break;
              }
            /* wait for btn release */
            while( !done )
              {
                /* increment counter */
                ez_SetCounter(btn, ptr->value + increment);
                XSync(dpy, False);

                if(--nnn > 0) EZ_SitForALittleBit(30000 * nnn);
                else EZ_SitForALittleBit(30000);

                while(XPending(dpy))
                  {
                    XNextEvent(dpy, &xevent);
                    if(xevent.type == ButtonRelease && xevent.xbutton.button == EZ_Btn1) done = 1;
                    if(xevent.type == Expose)
                      {
                        EZ_WidgetDispatchEvent(&xevent);
                        EZ_RemoveEvent(&xevent);
                      }
                    if(done) break;
                  }
              }
          }
      }
    break;
    case KeyPress:  /* may be ... */
      break;
    default:
      break;
    }
  xev->type = 0;  /* disable the default event handler */
}
/*********************************************************************/
static void freeCounter(EZ_Widget *w, void *data) { free(data);}
/*********************************************************************/
static EZ_Widget *CounterCreate(EZ_Widget *parent, char *iname, char *cname)
{
  ez_counter *record = (ez_counter *)malloc(sizeof(ez_counter));
  memset(record, 0, sizeof(ez_counter));

  record->value = 0; record->increment1 = 1;  record->increment2 = 10;

  record->frame = EZ_CreateWidgetXrm(EZ_WIDGET_FRAME,        parent,
                                     iname, cname,
                                     EZ_PADX,                0,
                                     EZ_PADY,                0,
                                     EZ_IPADX,               2,
                                     EZ_HEIGHT,              0,
                                     EZ_FILL_MODE,           EZ_FILL_BOTH,
                                     EZ_DESTROY_CALLBACK,    freeCounter, record,
                                     EZ_CLIENT_PTR_DATA,     record,
                                     0);
  record->btnLL =    EZ_CreateWidget(EZ_WIDGET_LABEL,        record->frame,
                                     EZ_ARROW_LABEL,         EZ_DOUBLE_LEFT_TRIANGLE,
                                     EZ_CLIENT_PTR_DATA,     record,
                                     EZ_CLIENT_INT_DATA,     -2,
                                     EZ_EVENT_HANDLER,       btnEHandle, record,
                                     EZ_BORDER_WIDTH,        2,
                                     EZ_BORDER_TYPE,         EZ_BORDER_RAISED,
                                     EZ_WIDTH,                0,
                                     0);
  record->btnL =     EZ_CreateWidget(EZ_WIDGET_LABEL,        record->frame,
                                     EZ_ARROW_LABEL,         EZ_LEFT_TRIANGLE,
                                     EZ_CLIENT_PTR_DATA,     record,
                                     EZ_CLIENT_INT_DATA,     -1,
                                     EZ_EVENT_HANDLER,       btnEHandle, record,
                                     EZ_BORDER_WIDTH,        2,
                                     EZ_BORDER_TYPE,         EZ_BORDER_RAISED,
                                     EZ_WIDTH,                0,
                                     0);
  record->label = EZ_CreateWidgetXrm(EZ_WIDGET_LABEL,        record->frame,
                                     "counter", "Counter",
                                     EZ_BORDER_TYPE,         EZ_BORDER_SUNKEN,
                                     EZ_BORDER_WIDTH,        2,
                                     EZ_WIDTH,               80,
                                     EZ_LABEL_POSITION,      EZ_CENTER,
                                     EZ_CLIENT_PTR_DATA,     record,
                                     EZ_PROPAGATE,           False,
```

```
                                     0);
  record->btnR =    EZ_CreateWidget(EZ_WIDGET_LABEL,        record->frame,
                                    EZ_ARROW_LABEL,          EZ_RIGHT_TRIANGLE,
                                    EZ_CLIENT_PTR_DATA,      record,
                                    EZ_CLIENT_INT_DATA,      1,
                                    EZ_EVENT_HANDLER,        btnEHandle, record,
                                    EZ_BORDER_WIDTH,         2,
                                    EZ_BORDER_TYPE,          EZ_BORDER_RAISED,
                                    EZ_WIDTH,                0,
                                    0);
  record->btnRR =   EZ_CreateWidget(EZ_WIDGET_LABEL,        record->frame,
                                    EZ_ARROW_LABEL,          EZ_DOUBLE_RIGHT_TRIANGLE,
                                    EZ_CLIENT_PTR_DATA,      record,
                                    EZ_CLIENT_INT_DATA,      2,
                                    EZ_EVENT_HANDLER,        btnEHandle, record,
                                    EZ_BORDER_WIDTH,         2,
                                    EZ_BORDER_TYPE,          EZ_BORDER_RAISED,
                                    EZ_WIDTH,                0,
                                    0);

  /* set up: parse resources 'value', 'increment' and 'counterWidth */
  {
    char str[64], *value; int a, b;
    if(EZ_GetWidgetResource(record->frame, "value", &value))
      if(sscanf(value, "%d", &a) == 1) record->value = a;
    if(EZ_GetWidgetResource(record->frame, "increment", &value))
      if(sscanf(value, "%d %d", &a, &b) == 2)
        { record->increment1 = a;  record->increment2 = b;}
    if(EZ_GetWidgetResource(record->frame, "counterWidth", &value))
      if(sscanf(value, "%d", &a) == 1 && a > 0)
        { EZ_ConfigureWidget(record->label, EZ_WIDTH,  a, 0); }

    sprintf(str, "%d", record->value);
    EZ_ConfigureWidget(record->label, EZ_LABEL_STRING, str, 0);
  }
  return(record->frame);
}
/*********************************************************************/
/* TEST */
main(int ac, char **av)
{
  EZ_Widget *counter;

  EZ_Initialize(ac,av,0);
  Counter_TYPE = EZ_RegisterCompositeWidget("counter", "Counter",
                              CounterCreate);
  counter  = EZ_CreateWidgetXrm(Counter_TYPE,    NULL,
                                 "counter1",      "Counter1",
                                 0);
  EZ_DisplayWidget(counter);
  EZ_EventMainLoop();
}
/*********************************************************************/
```

Chapter 7

Events

This chapter explains briefly the kinds of events *EZwgl* handles.

7.1 X Events

Internally, event handling is based on Xlib's event queuing mechanism. *EZwgl* does not queue events itself. It just reads events continuousely from Xlib's event queue and processes them when they arrive. In between, it checks and processes timer events and file events.

The following is the event mask used to create widget windows for most of the widgets in *EZwgl*.

```
KeyPressMask | ButtonReleaseMask | ExposureMask | ButtonPressMask |
StructureNotifyMask | ButtonMotionMask | EnterWindowMask |
LeaveWindowMask | PointerMotionMask | KeymapStateMask
```

In other words, most *EZ* widgets accept keyboard, button, pointer motion, enter/leave Window, exposure and window configuration events.

X events are handled by event handlers. Each widget has a default event handler. This event handler defines the default behavior of a widget in response to user inputs. If needed, one can register one or more private event handlers to a widget. The private event handers are invoked before the default event handler. A private event handler can disable the default event handler completely by using `EZ_RemoveEvent` after an event has been handled.

Event handlers are registered by

```
void EZ_AddEventHandler(EZ_Widget *widget, EZ_EventHandler hdlr,
                        void *data, int where);
typedef  void (*EZ_EventHandler)(EZ_Widget *widget, void *data,
                                 int etype, XEvent *event);
```

7.1.1 Write Simple Event Handlers

EZwgl keeps a few global variables to remember the mouse-button-state, pointer-position and last pressed-key. These global variables simplifies the complexity of writing simple event handlers. For sophisticated event handlers, unfortunately, one has to decipher XEvents.

Global Variables

- `int EZ_PointerCoordinates[2]` This variable holds the coordinate of the pointer relative to the upper-left corner of the widget window.
- `int EZ_PressedKey`. This variable holds the keycode translation of the last pressed key.
- `int EZ_ButtonState`. This variable holds current state of all mouse buttons. There are three macros associated with this variable. They report the state of a specific button.
  - `EZ_Button1Down()`
  - `EZ_Button2Down()`
  - `EZ_Button3Down()`

- `int EZ_MouseX, EZ_MouseY`. These two variable hold the current pointer location in an widget window. The position is measured relative to the **lower-left** corner of the widget window, offseted by the border width. These two variables are meant for 3DCanvas widgets, they are the screen coordinates of the pointer in the current viewport.

Symbolic Event Names

The next table lists the symbolic names for some of the X events the event dispatcher passes to private event handlers.

Symbolic name	Corresponding X event
EZ_BUTTON1_PRESS	ButtonPress, xbutton=Button1
EZ_BUTTON2_PRESS	ButtonPress, xbutton=Button2
EZ_BUTTON3_PRESS	ButtonPress, xbutton=Button3
EZ_BUTTON1_RELEASE	ButtonRelease, xbutton=Button1
EZ_BUTTON2_RELEASE	ButtonRelease, xbutton=Button2
EZ_BUTTON3_RELEASE	ButtonRelease, xbutton=Button3
EZ_REDRAW	Expose (for 3DCanvas only)
EZ_RESIZE	ConfigureNotify (for 3DCanvas only)
EZ_KEY_PRESS	KeyPress (see below)
EZ_ENTER_WINDOW	EnterNotify
EZ_LEAVE_WINDOW	LeaveNotify
EZ_POINTER_MOTION	MotionNotify

For `EZ_KEY_PRESS` events, the global variable `EZ_PressedKey` holds the keycode translation of the corresponding keysym. The next table lists the symbolic names of keycode translations for keysyms corresponding to function keys and other special keys.

Symbolic Name	Corresponding keysym
EZ_PAGE_UP_KEY	XK_Prior
EZ_PAGE_DOWN_KEY	XK_Next
EZ_HOME_KEY	XK_Home
EZ_LEFT_KEY	XK_Left or XK_KP_Left
EZ_RIGHT_KEY	XK_Right or XK_KP_Right
EZ_UP_KEY	XK_Up or XK_KP_Up
EZ_DOWN_KEY	XK_Down or XK_KP_Down
EZ_BEGIN_KEY	XK_Begin
EZ_END_KEY	XK_End
EZ_F1_KEY	XK_F1
EZ_F2_KEY	XK_F2
EZ_F3_KEY	XK_F3
EZ_F4_KEY	XK_F4
EZ_F5_KEY	XK_F5
EZ_F6_KEY	XK_F6
EZ_F7_KEY	XK_F7
EZ_F8_KEY	XK_F8
EZ_F9_KEY	XK_F9
EZ_F10_KEY	XK_F10
EZ_F11_KEY	XK_F11
EZ_F12_KEY	XK_F12

A Simple Event Handler

Listed below is a simple event handler. All it does is to display the event info.

```
#include "EZ.h"

static void sampleEHandler(EZ_Widget *widget, void *data, int etype, XEvent *xev)
{
  EZ_Widget *label = (EZ_Widget *)data;
  char str[256];

  switch(etype)
    {
    case EZ_BUTTON1_PRESS:
      sprintf(str, "Button1Press:  (x,y)=(%d, %d)",
              EZ_PointerCoordinates[0],EZ_PointerCoordinates[1]);
      break;
    case EZ_BUTTON1_RELEASE:
      sprintf(str, "Button1Release:  (x,y)=(%d, %d)",
              EZ_PointerCoordinates[0],EZ_PointerCoordinates[1]);
      break;
    case EZ_POINTER_MOTION:
      sprintf(str, "PointerMotion:  (x,y)=(%d, %d)",
              EZ_PointerCoordinates[0],EZ_PointerCoordinates[1]);
      break;
    case EZ_KEY_PRESS:
      sprintf(str,"KeyPress: key=%c, keycode=%d", EZ_PressedKey,EZ_PressedKey);
      break;
    case EZ_LEAVE_WINDOW:
      sprintf(str, "LeaveWindow");
      break;
    case EZ_ENTER_WINDOW:
      sprintf(str, "EnterWindow");
```

```
      break;
    case EZ_REDRAW:
      sprintf(str, "Redraw");
      break;
    default:
      str[0]=0;
      break;
    }
  if(str[0]) EZ_ConfigureWidget(label, EZ_LABEL_STRING, str, 0);
}

main(int ac, char **av)
{
  EZ_Widget *frame, *label;

  EZ_Initialize(ac,av,0);

  frame = EZ_CreateWidget(EZ_WIDGET_FRAME, NULL,
                          EZ_ORIENTATION, EZ_VERTICAL_BOTTOM,
                          EZ_SIZE,        400, 300,
                          0);
  label = EZ_CreateWidget(EZ_WIDGET_LABEL, frame,
                          EZ_EXPAND,       True,
                          EZ_PROPAGATE,    False,
                          EZ_BORDER_WIDTH, 2,
                          EZ_BORDER_TYPE,  EZ_BORDER_SUNKEN,
                          EZ_TEXT_LINE_LENGTH, 80,
                          EZ_FOREGROUND, "red",
                          EZ_BACKGROUND, "bisque2",
                          0);

  EZ_AddEventHandler(frame, sampleEventHandler, label, 0);

  EZ_DisplayWidget(frame);
  EZ_EventMainLoop();
}
/************************************* ExampleEHandler ****************************/
```

Enter Events

EZwgl provides a command for sending events to widgets.

```
void EZ_EnterEvent(EZ_Widget *widget, int eType, int *values);
```

This function enters the specified event. `etype` must be one of
`EZ_REDRAW`, `EZ_RESIZE`, `EZ_BUTTON1_PRESS`, `EZ_BUTTON2_PRESS`, `EZ_BUTTON3_PRESS`, `EZ_BUTTON1_RELEASE`, `EZ_BUTTON2_RELEASE`, `EZ_BUTTON3_RELEASE`, `EZ_POINTER_MOTION`, `EZ_ENTER_WINDOW`, `EZ_LEAVE_WINDOW` and `EZ_KEY_PRESS`.

For events of type `EZ_BUTTON?_PRESS`, `EZ_BUTTON?_RELEASE` or `EZ_POINTER_MOTION`, `values`, if not `NULL` is an array of two integers which specifies the pointer location. For `EZ_KEY_PRESS` event, `values`, if not `NULL` is a pointer to an integer which specifies the keycode for the key that is being pressed.

Care must be taken if you send events to a composite widget. You have to send the event to the right internal widget in order to have it handled.

This function behave differently for 3DCanvas widget. See `EZ_GLEnterEvent`.

7.2 Timer Events

There are two lists of timers in the *EZwgl*, a list of repeating timers and a list of one-time timers. The second list is an ordered list sorted according to the timeouts while the first one is not sorted.

Timers are created by

```
EZ_Timer *EZ_CreateTimer(long sec, long usec, int repetition,
          EZ_CallBack callback, void *pdata, int idata);
```

This function creates a timer and returns the allocated timer data structure. If `repetition` is 0, the timer created is put in the one-time timer list and will be fired exactly once if at all. If `repetition` is nonzero, the timer is inserted into the repeating timer list and it will be fired exactly `repetition` times or indefinitely depending on whether `repetition` is positive or not.

The following three functions can be used to modify the state of an existing timer.

```
EZ_Timer *EZ_ResetTimer(EZ_Timer *timer, long sec, long usec,
                        int repetition, EZ_CallBack callback,
                        void *pdata, int idata);
```

```
EZ_Timer *EZ_RestartTimer(EZ_Timer *timer);
void      EZ_CancelTimer(EZ_Timer *timer);
```

Here are a few more interface functions.

`void EZ_SetTimerPtrData(EZ_Timer *timer, void *data);`
attach client pointer data to a timer.

`void EZ_SetTimerIntData(EZ_Timer *timer, int value);`
attach an integer to a timer.

`void EZ_SetTimerClientData(EZ_Timer *timer, void *p, int i);`
attach client data to a timer.

`void EZ_SetTimerCallBack(EZ_Timer *timer, EZ_CallBack cb, void *data);`
set timer callback procedure.

`void EZ_GetTimerState(EZ_Timer *timer, long *sec, long *usec, int *repetition);`
return the current state of a timer.

`void  EZ_GetTimerPtrData(EZ_Timer *timer,void **dat_return);`
return the client data attached to a timer.

`int EZ_GetTimerdIntData(EZ_Timer *timer);`
return the integer attached to a timer.

`void EZ_GetTimerdClientData(EZ_Timer *timer, void **p, int *i);`
return the pointer and the integer attached to a timer.

7.2.1 An Example

```
/************************* ExampleTimer ***********************/
#include "EZ.h"

static void rtimer_callback(EZ_Timer *timer, void *pdata)
{
  long delay_sec, delay_usec;
  int  n_calls_remaining;
  char str[256];

  EZ_GetTimerState(timer, &delay_sec, &delay_usec, &n_calls_remaining);
  if(n_calls_remaining == 0) {EZ_Shutdown(); exit(0);}
  sprintf(str,"%.1f seconds remaining before destruction",
          (float) (delay_usec * n_calls_remaining)/1000000.0);
  EZ_ConfigureWidget((EZ_Widget*)pdata, EZ_LABEL_STRING, str, 0);
}

main(int ac, char **av)
{
  EZ_Widget *frame, *label;
  EZ_Timer  *RTimer;

  EZ_Initialize(ac,av,0);
  frame = EZ_CreateWidget(EZ_WIDGET_FRAME, NULL,
                          EZ_LABEL_STRING, "A Timer",
                          0);
  label = EZ_CreateWidget(EZ_WIDGET_NORMAL_BUTTON, frame,
                          EZ_FOREGROUND,  "red",
                          EZ_PROPAGATE,    0,
                          EZ_LABEL_STRING, "20.0 seconds remaining before destruction",
                          EZ_LABEL_JUSTIFICATION, EZ_CENTER,
                          0);
  RTimer = EZ_CreateTimer(0, 100000, 200, rtimer_callback, label, 0);
  EZ_DisplayWidget(frame);
  EZ_EventMainLoop();
}
```

7.3 File Events

EZwgl provides a timer-like mechanism for reading and writing files in a event-driven fashion.

To queue file events, use

`EZ_Input *EZ_AddInput(int fd, int mask, EZ_InputCallBack callback, void *data);`

This function registers a new source of events: the status change of the specifed file descripter. From the time a `fd` is registered on, the specified status of `fd` is watched by the main event dispatcher. Whenever the specified status changes, `callback` is invoked.

Here `fd` specifies a file descriptor. `mask` specifies what status to watch for. It is an or'ed combination of `EZ_READABLE_MASK`, `EZ_WRITABLE_MASK` and `EZ_EXCEPTION_MASK`. `callback` spcifies a procedure. It will be invoked whenever the specified status of `fd` changes. `data` specifies an arbitary client data to be passed to `callback` when invoked.

`EZ_InputCallBack` is a procedure of the following type.

```
void inputcallback(EZ_Input *obj, void *data, int fd, int mask)
```

To remove file events, use

```
void EZ_RemoveInput(EZ_Input inputId);
```

7.3.1 An Example

In this example, we watch the readable status of stdin, if there are anything to read, we read up to 1024 characters a time and append the string at the end of the text widget.

```
/****************** ExampleInput *************************************/
#include <unistd.h>

#include "EZ.h"
static void readStdin(EZ_Input *id, void *data, int fd, int mask)
{
  EZ_Widget *tw = (EZ_Widget *)data;
  char buf[1024];
  int n;
  if(mask & EZ_READABLE_MASK)
    {
      n = read(fd, buf, 1023);
      if(n > 0)
        {
          buf[n] = 0;
          EZ_TextEndOfBuffer(tw);
          EZ_TextInsertString(tw,buf);
        }
    }
}

main(int ac, char **av)
{
  EZ_Widget *textW;

  EZ_Initialize(ac,av,0);

  textW = EZ_CreateWidget(EZ_WIDGET_TEXT, frame,
                          EZ_TEXT_BACKGROUND, "white",
                          EZ_BORDER_WIDTH,     1,
                          EZ_SIZE,             300, 300,
                          0);

  EZ_DisplayWidget(textW);
  EZ_AddInput(0, EZ_READABLE_MASK, readStdin, textW);
  EZ_EventMainLoop();
}
/****************** ExampleInput *************************************/
```

7.4 Idle Callbacks

Idle callbacks are procedures executed at the time when there are no user inputs to process. Use the following three functions to manage idle callbacks.

```
void     EZ_AddIdleCallBack(EZ_CallBack cb, void *data);
void     EZ_RemoveIdleCallBack(EZ_CallBack cb, void *data);
void     EZ_RemoveAllIdleCallBack(void);
```

One registered, an idle callback will be executed repeatly untile it is removed. To ensure an idle callback to be executed exactly once, remove the callback itself inside the callback.

If multiple idle callbacks are registered, *EZwgl* executes them in turn.

7.5 ClientMessage Handler

The *EZ* widget library does not handle ClientMessage events in any useful way. The default behavior is that the event dispatcher drops all ClientMessage events except one, a message with `message_type` `WM_PROTOCOLS` and content `XA_WM_DELETE_WINDOW` from your window manager. (It is sent from your window manager when you close the window.) For this message, the default action is to exit, which is not desirable for applications that uses more than of toplevel widgets. For example, in a file manager, closing a directory lister should not exist the application.

The *EZ* widget library includes a command for applications to hook in their own clientMessages handler.

```
void EZ_SetClientMessageHandler(EZ_EventHandler handler,
                                void *data);
```

This function overides the default ClientMessage handler. If set succesfully, `handler` will be invokde whenever the application receives a ClientMessage event.

Set a `NULL` ClientMessage handler reverts to the default handler.

Example: A ClientMessage handler

Here is an example that uses a private ClientMessage handler.

When you close the main window from the menu, it pops up a dialogue asking for confirmation.

```
/************************* Example **********************/
#include "EZ.h"

EZ_Widget *Dialogue;        /* a dialogue       */
Atom       WMProtocolsAtom;  /* WM_PROTOCOLS     */
Atom       DeleteWindowAtom; /* WM_DELETE_WINDOW */

void yesCallBack(EZ_Widget *widget, void *data)
{ EZ_Shutdown();  exit(0); }

void noCallBack(EZ_Widget *widget, void *data)
{
  if(widget)
    {
      EZ_Widget *toplevel = (EZ_Widget *)data;
      EZ_HideWidget(toplevel);
      EZ_ReleaseGrab();
    }
}

void buttonCallBack(EZ_Widget *widget, void *data)
{
  EZ_DisplayWidget(Dialogue);
  EZ_SetGrab(Dialogue);
  EZ_SetFocusTo((EZ_Widget *)EZ_GetWidgetPtrData(Dialogue));
}

void clientMessageHandler(EZ_Widget *widget, void *data, int etype, XEvent *xev)
{
  XClientMessageEvent *ev = (XClientMessageEvent *)xev;

  if(ev->message_type == WMProtocolsAtom)
    {
      Atom c = (ev->data.l)[0];
      if(c == DeleteWindowAtom)
{
  EZ_DisplayWidget(Dialogue);
  EZ_SetGrab(Dialogue);
  EZ_SetFocusTo((EZ_Widget *)EZ_GetWidgetPtrData(Dialogue));
}
    }
}

main(int argc, char **argv)
{
  EZ_Widget *frame, *button, *buttonA;
  EZ_Widget *yes, *no;

  EZ_Initialize(argc,argv,0);
  DeleteWindowAtom = EZ_GetAtom("WM_DELETE_WINDOW");
  WMProtocolsAtom = EZ_GetAtom("WM_PROTOCOLS");

  frame   = EZ_CreateWidget(EZ_WIDGET_FRAME,          NULL,
                            0);
  button  = EZ_CreateWidget(EZ_WIDGET_NORMAL_BUTTON,  frame,
                            EZ_LABEL_STRING,          "Press me",
                            EZ_CALLBACK,              buttonCallBack, NULL,
                            0);
  buttonA = EZ_CreateWidget(EZ_WIDGET_NORMAL_BUTTON,  frame,
                            EZ_LABEL_STRING,          "A Button",
                            0);
  Dialogue= EZ_CreateWidget(EZ_WIDGET_FRAME,          NULL,
                            EZ_LABEL_STRING,          "Want to Quit?",
                            EZ_TRANSIENT,             True,
                            0);
  yes     = EZ_CreateWidget(EZ_WIDGET_NORMAL_BUTTON,  Dialogue,
```

```
                           EZ_LABEL_STRING,          "Yes",
                           EZ_CALLBACK,              yesCallBack,NULL,
                           0);
  no      = EZ_CreateWidget(EZ_WIDGET_NORMAL_BUTTON,  Dialogue,
                           EZ_LABEL_STRING,          "No",
                           EZ_CALLBACK, noCallBack,  Dialogue,
                           0);
  EZ_SetWidgetPtrData(Dialogue, no);
  EZ_SetClientMessageHandler(clientMessageHandler, NULL);
  EZ_DisplayWidget(frame);
  EZ_EventMainLoop();
}
/************************** Example **********************/
```

Chapter 8

Drag and Drop

EZwgl supports drag and drop. Most widgets and display items are potentially drag sources and/or drop targets. An object becomes a drag source if it has one or more DnD encoders registered to it. An object becomes a drop target if it has one or more DnD decoders registered.

DnD encoders and DnD decoders are procedures of the following prototypes.

```
int encoder(void *object, void *data, char **message, int *length, int *needFree)
int decoder(void *object, void *data, char *message,  int length)
```

When invoked by *EZwgl*, `object` is the widget or display item the procedure is registered to. `data` is the client data the application has told *EZwgl* to pass back when invoked. For encoders, `message` returns the encoded message, `length` returns the length of message and `needFree` returns a boolean value signals whether *EZwgl* should free the encoded message when done with it. For decoders, `message` and `length` are the message and the length of the message passed by *EZwgl*. The message will be automatically freed by *EZwgl* when the relevent DnD transaction finishes. Both encoders and decoders should return `EZ_DND_SUCCESS` if conversion succeed and return `EZ_DND_FAILUER` when conversion fails.

You can only transfer data in 8-bit quantities using DnD. If you need to transfer information of other kinds, for example, 4 bytes quantities, you'll have to engineer your encoder and decoder to convert the quantity to and from 8-bit quantities.

8.1 How Drag and Drop Works

Each *EZwgl* application has an internal window (unmapped) for communication. Drag and Drop transactions are carried out by changing the window properties of these communication windows.

For each type of information an application transfers, there is an unique internal identifier (an **Atom**) that specifies the information type. This identifier is called a *conversion target*. It is the responsibility of an application to register the conversion targets before using them. Convertion targets are created by `XInternAtom` or `EZ_GetAtom`. These routines register a name to the X server. Once created, an atom remains unique till server shutdown.

The right mouse button (Button3) is the drag and drop button for *EZwgl*. When a user press-and-drag Button3 over a drag source, *EZwgl* initiates a new DnD transaction, executes the initialization encoder (registered to the special target `EZ_DND_DRAG_INIT_ATOM`), build a list of conversion targets the drag source converts, and broadcast to all *EZwgl* applications this list of targets. It then replace the default cursor by the drag cursor if the drag source has one, or a snapshot of the drag source will be made to follow the pointer, or if a drag icon is supplied, it pops out the drag icon. Whenever the pointer moves, the drag source broadcast its position to all *EZwgl* applications. If the drag is over a potential drop target, the border of the potential target lights up. When the drag is released over a drop target, the following sequence of actions will be initiated.

1. The drag source invokes its drag setup encoder (a special encoder registered to the special target `EZ_DND_DRAG_START_ATOM`) and its callbacks; and then sends a message containing the setup message and the list of targets the source converts to the drop target.

2. The target examine the list of targets the source supplied and examine the list of decoders registered to itself, bulids a list of matching targets both the source and the target convert and invokes its drop setup decoder (a special decoder registered to the special target `EZ_DND_DROP_START_ATOM`) and its callbacks.

3. If there are no matching targets on the matching list, the target aborts its drop operation, invokes its drag clean up decoder (a decoder registered to the special target `EZ_DND_DROP_FINISH_ATOM`) and its callback, and sends a message to the drag source to abort. Otherwise, it sends a convertion request, start with the first available matching target.

4. Upon reciving a conversion request, the drag source invokes its relevent encoder to do the conversion. If succesful, the encoded message is send to the target. Otherwise, it sends a message to the drop target that conversion for the requested target has failed.

5. If the drop target receives a conversion failed message, it examine the list of matching targets. If there are more targets on the matching list, it starts step 3 again. Otherwise, It sends a message to the drag source to cancell the current drag, invokes its drop clean up procedures and aborts the current DnD transaction itself.

   If the drop target receives a conversion, it invokes its relevent DnD decoder to decode the message. If succesfull, it sends an acknowledgement message to the drag source and, if there is a callback for this decoder, it invokes the callback. If decoding fails, it'll try the next target on the matching list and start step 3 again.

6. If the drag source receives an acknowledgement message, it invokes the callback for the current encoder if there is one; it then invokes the drag clean up procedures (an encoder registered to the special target `EZ_DND_DRAG_FINISH_ATOM`) and marks the current DnD transaction complete. If it receives a message to abort, it invokes the drag clean up procedures and aborts the current DnD transaction.

8.2 DnD Setup and Clean Up Procedures

EZwgl reserves five special convertion targets:

```
EZ_DND_DRAG_INIT_ATOM,
EZ_DND_DRAG_START_ATOM, EZ_DND_DRAG_FINISH_ATOM,
EZ_DND_DROP_START_ATOM, EZ_DND_DROP_FINISH_ATOM
```

They are global atoms set by *EZwgl* at the initialization time. Encoders/decoders registered to these targets are treated as setup/clean up procedures for DnD.

Encoder for target `EZ_DND_DRAG_INIT_ATOM` is the drag initialization encoder. This encoder and its callback are invoked when the user press-and-move the DnD mouse button over a drag source. The encoder behaves like a callback. Its main purpose is to setup a drag icon. It should not do any conversions as the encoded message will never be used.

Encoder for target `EZ_DND_DRAG_START_ATOM` is the drag setup encoder. This encoder is invoked when the drag is droped at a drop site. The encoded message will be the first message sent to the drop site, along with a list of targets the source converts. Drag setup actions are in the following order.

- Build the list of targets it converts.
- Invoke setup encoder.
- Invoke setup encoder callback.
- Send the encoded message and the list of targets to the drop site.

The list of targets is available to the setup encoder and its callback. One can get this list by calling

```
void EZ_DnDGetDragConversionTargets(Atom **targets_ret,
                                    int **ntargets_ret,
                                    unsigned int **action_tag)
```

The setup encoder and its callback are free to modify the list of targets. For example, the callback may popup a dialogue, ask the user for some infomation and select one or more targets based on the response.

Encoder for target `EZ_DND_DRAG_FINISH_ATOM` is the drag cleanup encoder. This encoder is invoked whenever a DnD transaction is deemed complete, success or not. The encoded message is never used.

Decoder for target `EZ_DND_DROP_START_ATOM` is the the drop setup decoder. It is invoked right after the drop site has received the initial drag setup message (list of targets plus possiblly a short message) from the drag source. The order of actions at the setup time are:

- Build a list of matching targets it decodes.
- Invoke the setup decoder to decode the setup message sent by the drag source.
- Invoke the callback of the setup decoder.

The list of matching decoders is available for both the decoder and its callback. One can get this list by calling

```
void EZ_DnDGetMatchedConversionTargets(Atom **targets_ret,
                                       int **ntargets_ret,
                                       unsigned int **action_tag_ret)
```

The decoder and its callback are free to modify this list.

Encoder for target `EZ_DND_DROP_FINISH_ATOM` is the drop cleanup decoder. This decoder is invoked whenever a DnD transaction is deemed complete, success or not.

8.3 A DnD Example

We now present a simple example to demonstrate you the mechanics for implementing drag and drop. We implement two short programs, the first one contains a few buttons which serve as drag sources and the second one contains a label that serves as a drop target. Our conversion target will be `MY_BG_ATOM`, which stands for background color. For each button, we register a DnD encoder which encode the background color of the button to a string. For the label, we register a DnD decoder which decodes the conversion (actually just uses it) and use the result to set the background color of the label.

```
/************************ Example Source ***********************************/
#include "EZ.h"

int encodeBG(EZ_Widget *, void *, char **, int *, int *);
int decodeBG(EZ_Widget *, void *, char *, int);

static char *colors[] = { "red", "green", "blue", "cyan",
                          "magenta", "yellow", "red3","#cf00cf"};

Atom MY_BG_ATOM;

main(int ac, char **av)
{
  EZ_Widget *frame, *tmp, *btn;
  int i;

  EZ_Initialize(ac,av,0);
  MY_BG_ATOM = EZ_GetAtom("MY_BG_ATOM");

  frame = EZ_CreateWidget(EZ_WIDGET_FRAME, NULL,
                          EZ_LABEL_STRING, "Drag sources",  0);
  tmp  = EZ_CreateWidget(EZ_WIDGET_FRAME, frame,
                         EZ_ORIENTATION, EZ_VERTICAL,
                         EZ_FILL_MODE, EZ_FILL_BOTH,  0);
  for(i = 0; i < 8; i++)
    {
      btn = EZ_CreateWidget(EZ_WIDGET_NORMAL_BUTTON, tmp,
                            EZ_LABEL_STRING, colors[i],
                            EZ_EXPAND, True,
                            EZ_BACKGROUND, colors[i],
                            EZ_DND_BUBBLE_HELP, "DnD bubble help",
                            0);
      EZ_WidgetAddDnDDataEncoder(btn, MY_BG_ATOM, 0,
                                 encodeBG, NULL,
                                 NULL, NULL);
      EZ_WidgetAddDnDDataDecoder(btn, MY_BG_ATOM, 0,
                                 decodeBG, NULL,
                                 NULL, NULL);
    }

  EZ_DisplayWidget(frame);
  EZ_EventMainLoop();
}

int encodeBG(EZ_Widget *widget, void *data,
             char **message, int *length, int *needFree)
{
  static char str[32];
  if(widget)
    {
      unsigned long pv;
      int r,g,b;

      /* why not just use the label string ?*/
      EZ_GetBackgroundPV(widget, &pv, 0, 0);
      EZ_PixelValue2RGB(pv, &r, &g, &b);
      sprintf(str, "#%02x%02x%02x", r, g, b);

      *length = strlen(str);
      *message = str;
      *needFree = 0;

      return(EZ_DND_SUCCESS);
    }
  return(EZ_DND_FAILURE);
}

int decodeBG(EZ_Widget *widget, void *data,
             char *message, int length)
{
  if(widget)
    {
      /* the message is a correct color specification */
      EZ_ConfigureWidget(widget, EZ_FOREGROUND, message, 0);
      return(EZ_DND_SUCCESS);
    }
  return(EZ_DND_FAILURE);
}

/************************ Example Target ***********************************/
```

```
#include "EZ.h"

int decodeBG(EZ_Widget *, void *, char *, int);

Atom MY_BG_ATOM;

main(int ac, char **av)
{
  EZ_Widget *frame, *label;

  EZ_Initialize(ac,av,0);
  MY_BG_ATOM = EZ_GetAtom("MY_BG_ATOM");

  frame = EZ_CreateWidget(EZ_WIDGET_FRAME,NULL,
                          EZ_LABEL_STRING, "Drag target", 0);
  label = EZ_CreateWidget(EZ_WIDGET_LABEL, frame,
                          EZ_LABEL_STRING, "Drag and drop a color button to change my FG color",
                          EZ_DND_BUBBLE_HELP, "DnD bubble help", 0);
  EZ_WidgetAddDnDDataDecoder(label, MY_BG_ATOM, 0,
                             decodeBG, NULL,
                             NULL, NULL);
  EZ_DisplayWidget(frame);
  EZ_EventMainLoop();
}

int decodeBG(EZ_Widget *widget, void *data,
             char *message, int length)
{
  if(widget)
    {
      /* the message is a correct color specification */
      EZ_ConfigureWidget(widget, EZ_BACKGROUND, message, 0);
      return(EZ_DND_SUCCESS);
    }
  return(EZ_DND_FAILURE);
}
```

8.4 Related Routines

```
void EZ_WidgetAddDnDDataEncoder(EZ_Widget *widget, Atom target,
                                int tag, EZ_DnDEncoder encoder, void *edata,
                                EZ_CallBack callback, void *cdata);
void EZ_WidgetDeleteDnDDataEncoder(EZ_Widget *widget,
                    EZ_DnDEncoder encoder, void *edata);
void EZ_WidgetDeleteAllDnDDataEncoders(EZ_Widget *widget);

void EZ_ItemAddDnDDataEncoder(EZ_Item *item, Atom target,
                              int tag, EZ_DnDEncoder encoder, void *edata,
                              EZ_CallBack callback, void *cdata);
void EZ_ItemDeleteDnDDataEncoder(EZ_Item *item,
                             EZ_DnDEncoder encoder, void *edata);
void EZ_ItemDeleteAllDnDDataEncoders(EZ_Item *item);
```

The tag argument used in `*Add*Encoder` is used to choose conversion methods. It is useful for the following situations.

- A drag source has multiple encoders for the same conversion target. For example, when transfering files, one may want either to copy a file or move a file to the destination specified by the drop site. This can be achieved by registering multiple encoders with different tags. For example, shift-press-drag-relase for copying the file and cntrl-press-drag-release for moving the file.

- Sometimes it is preferable to enforce a more strict drag and drop policy so that a careless drag and drop will not do any harm. For example, if you write a file manager and want to use DnD to remove files, it may be better to enforce a control-press-drag-release policy for the trash can.

- The tag value `~0` is reserved for a special purpose: drop at the root window of your display. Strictly speaking, the root window is not a valid drop site because it does not talk to the drag source. Nevertheless, for applications like file managers, it is useful to have a feature so a user can drop an item at the root window. Since the root window is not an official drop site, there will be no data transfers. The action is completely handled by the drag source. If you want to use this feature. Here are the tricks.

  1. Write a special encoder which does not convert anything, but rather, acts like a callback. In the example of droping a file item over the background, you may want to create a Icon widget, set it up properly and display

it. The encoder has to return `EZ_DND_SUCCESS` on success and `EZ_DND_FAILURE` on failure. The callbacks for this special encoder behave exactly the same as that of a regular encoder. It is invoked when the encoder returns `EZ_DND_SUCCESS`.

2. Register your special encoder with a tag value `~0`.

```
void EZ_WidgetAddDnDDataDecoder(EZ_Widget *widget, Atom target,
                      int tag, EZ_DnDDecoder decoder, void *edata,
                      EZ_CallBack callback, void *cdata);
void EZ_WidgetDeleteDnDDataDecoder(EZ_Widget *widget,
                      EZ_DnDDecoder decoder, void *data);
void EZ_WidgetDeleteAllDnDDataDecoders(EZ_Widget *widget);

void EZ_ItemAddDnDDataDecoder(EZ_Item *item, Atom target,
                     int tag, EZ_DnDDecoder decoder, void *edata,
                     EZ_CallBack callback, void *cdata);
void EZ_ItemDeleteDnDDataDecoder(EZ_Item *item,
                     EZ_DnDDecoder decoder, void *data);
void EZ_ItemDeleteAllDnDDataDecoders(EZ_Item *item);

void EZ_GetDnDDropLocation(int *rootX, int *rootY,
                           int *winX, int *winY);
```

8.5 Another DnD Example

This section explains you through an example the steps needed for implementing drag and drop. The example is about using drag and drop to transfer files.

We use a ListTree widget to list directories and files. The file items will be the drag sources. We use a TextWidget for a drop target. When the user press-drag-release a file item over the window of TextWidget, the file will be loaded into the text widget.

How do we transfere files? The simplest way is to transfer the complete path of files. This will work if the file-lister and text-widget are running under the same environment. It will probabily not work if, say, the file-lister and the text-widget are running on machines across two different local networks. So we'll need a way to convert the contents of a file, just in case the text-widget cannot find the file the file-lister supplied.

So we need two target items. We have to create them first.

```
  MY_FILE_NAME_ATOM      = EZ_GetAtom("MY_FILE_NAME_ATOM");
  MY_FILE_CONTENTS_ATOM  = EZ_GetAtom("MY_FILE_CONTENTS_ATOM");
```

Now we come to the most important part, writing conversion routines. We first explain the encoders which will be registered to file-items. For each file-item, we'll record its full path and the length of the path in the client data slots of its private data structure. So encode the path name does not need too much work.

```
int encodeFileName(EZ_Item *item, void *data,
                   char **message, int *length, int *needFree)
{
  if(item)
    {
      char *ptr = (char *)EZ_GetItemPtrData(item);
      int  len = EZ_GetItemIntData(item);
      if(len > 0)
        {
          *length = len;
          *message = ptr;
          *needFree = 0;   /* we don't want to free it */
          return(EZ_DND_SUCCESS);
        }
    }
  return(EZ_DND_FAILURE);
}
```

To encode the contents of a file, we'll have to figure out the length of the file and allocate a buffer to copy the the contents. We'll do this naively.

```
int encodeFileContents(EZ_Item *item, void *data,
                       char **message, int *length, int *needFree)
{
  if(item)
    {
      char *ptr = (char *)EZ_GetItemPtrData(item);
```

```
      int  len = EZ_GetItemIntData(item);
      if(len > 0)
        {
          char *msg;
          int  c, totalLength = 0;
          FILE *fp = fopen(ptr, "r");
          if(fp)
           {
              while(fgetc(fp) != EOF) totalLength++;
              (void)rewind(fp);
              msg = (char *)malloc( (totalLength + 1)*sizeof(char));
              ptr = msg;
              while((c = fgetc(fp)) != EOF) *ptr++ =c;
              fclose(fp);
             *length = totalLength;
             *message = msg;
             *needFree = 1; /* ask EZWGL to free msg when done */
             return(EZ_DND_SUCCESS);
          }
        }
    }
  return(EZ_DND_FAILURE);
}
```

Now the decoders. They are registered to the text-widget. For file names, we'll try to open the file for reading. If succesful, we'll call `EZ_TextLoadFile` to load the file into the text-widget. Otherwise, we'll return `EZ_DND_FAILURE`.

```
int decodeFileName(EZ_Widget *widget, void *data,
                   char *message, int length)
{
  if(widget)
    {
      if(length > 0)
        {
          FILE *fp = fopen(message, "r");
          if(fp)
            {
              fclose(fp);
              EZ_TextLoadFile(widget, message);
              return(EZ_DND_SUCCESS);
            }
         }
    }
  return(EZ_DND_FAILURE);
}
```

To decode the file contents, we'll just treat it as a string and call `EZ_TextInsertString` to insert it into the text-widget.

```
int decodeFileContents(EZ_Widget *widget, void *data,
                       char *message, int length)
{
  if(widget)
    {
      if(length > 0)
        {
          EZ_FreezeWidget(widget);
          EZ_TextClear(widget);
          EZ_TextInsertString(widget, message);
          EZ_UnFreezeWidget(widget);
          return(EZ_DND_SUCCESS);
        }
    }
  return(EZ_DND_FAILURE);
}
```

We have done the essential part needed for drag and drop. Now we'll just have to put things together and register the encoders and decoders. Unfortunately the order of registration matters. The *EZwgl* will try the conversion targets from the first to the last of the target list registered to an object. So we'll register the (en)decoder for target `MY_FILE_NAME_ATOM` first, since converting file names does not need too much data movement. It is the prefered target.

List below is the complete code for this example.

```
/****************** Example Source *************************************/
#include "EZ.h"

int     encodeFileName(EZ_Widget *, void *, char **, int *, int *);
int     encodeFileContents(EZ_Widget *, void *, char **, int *, int *);
void    destroyCallBack(EZ_Item *, void *);

EZ_TreeNode *MyCreateFileItem(char *);

Atom MY_FILE_NAME_ATOM;
Atom MY_FILE_CONTENTS_ATOM;

main(int ac, char **av)
{
```

```
  EZ_Widget *frame, *listTree;
  EZ_TreeNode *root;

  EZ_Initialize(ac,av,0);

  /* the two target atoms */
  MY_FILE_NAME_ATOM = EZ_GetAtom("MY_FILE_NAME_ATOM");
  MY_FILE_CONTENTS_ATOM = EZ_GetAtom("MY_FILE_CONTENTS_ATOM");

  frame = EZ_CreateWidget(EZ_WIDGET_FRAME, NULL,
                          EZ_LABEL_STRING, "Drag sources",
                          EZ_FILL_MODE, EZ_FILL_BOTH,
          EZ_WIDTH, 300, EZ_HEIGHT, 400, 0);
  listTree = EZ_CreateWidget(EZ_WIDGET_TREE, frame, 0);

  /* we have to use a customized fileNode creater
   * to register DnD encoders and to remember pathnames
   */
  (void)EZ_SetDirTreeFileNodeCreator(MyCreateFileItem);
  root = EZ_CreateDirTree("./*"); /* */
  EZ_SetListTreeWidgetTree(listTree, root);

  EZ_DisplayWidget(frame);
  EZ_EventMainLoop();
}

EZ_TreeNode *MyCreateFileItem(char *fname)
{
  EZ_TreeNode *node = NULL;
  if(fname)
    {
      EZ_Item *item = NULL;
      char *ptr, name[256];
      int len;

      /* don't mess with fname, work on a local copy of it */
      strcpy(name, fname);
      len = strlen(name);
      if(name[len-1] == '/') name[len-1] = 0;
      if((ptr = strrchr(name,'/'))) ptr++;
      else ptr = name;

      item = EZ_CreateLabelItem(ptr, NULL);
      if(item)
         {
          /* store the full pathname. Use destroyCallback to
           * release the allocated memory !!
           */
          char *path = (char *)malloc((len+2)*sizeof(char));
          (void)strcpy(path,fname);
          EZ_ConfigureItem(item, EZ_CLIENT_PTR_DATA, path,
                           EZ_CLIENT_INT_DATA, len,
                           EZ_DESTROY_CALLBACK, destroyCallBack, path, 0);
          EZ_ItemAddDnDDataEncoder(item,MY_FILE_NAME_ATOM,0,
                                   encodeFileName,NULL, NULL, NULL);
          EZ_ItemAddDnDDataEncoder(item,MY_FILE_CONTENTS_ATOM,0,
                                   encodeFileContents, NULL, NULL, NULL);
          node = EZ_CreateTreeNode(NULL, item);
        }
    }
  return(node);
}

void  destroyCallBack(EZ_Item *item, void *data)
{
  if(data) (void)free((char *)data); /* free the allocated path */
}

int encodeFileName(EZ_Item *item, void *data,
                   char **message, int *length, int *needFree)
{
  if(item)
    {
      char *ptr = (char *)EZ_GetItemPtrData(item);
      int  len = EZ_GetItemIntData(item);
      if(len > 0)
        {
          *length = len;
          *message = ptr;
          *needFree = 0;
          return(EZ_DND_SUCCESS);
        }
    }
  return(EZ_DND_FAILURE);
}

int encodeFileContents(EZ_Item *item, void *data,
                       char **message, int *length, int *needFree)
{
  if(item)
```

```
    {
      char *ptr = (char *)EZ_GetItemPtrData(item);
      int  len = EZ_GetItemIntData(item);
      if(len > 0)
        {
          char *msg;
          int  c, totalLength = 0;
          FILE *fp = fopen(ptr, "r");
          if(fp) while(fgetc(fp) != EOF) totalLength++;
          (void)fseek(fp, 0L, SEEK_SET);
          msg = (char *)malloc( (totalLength + 1)*sizeof(char));
          ptr = msg;
          while((c = fgetc(fp)) != EOF) *ptr++ =c;
          fclose(fp);
         *length = totalLength;
         *message = msg;
         *needFree = 1;
         return(EZ_DND_SUCCESS);
        }
    }
  return(EZ_DND_FAILURE);
}

/****************** Example Target **************************************/
#include "EZ.h"

int decodeFileName(EZ_Widget *, void *, char *, int);
int decodeFileContents(EZ_Widget *, void *, char *, int);

Atom MY_FILE_NAME_ATOM;
Atom MY_FILE_CONTENTS_ATOM;

main(int ac, char **av)
{
  EZ_Widget *frame, *textW;

  EZ_Initialize(ac,av,0);

  MY_FILE_NAME_ATOM = EZ_GetAtom("MY_FILE_NAME_ATOM");
  MY_FILE_CONTENTS_ATOM = EZ_GetAtom("MY_FILE_CONTENTS_ATOM");


  frame = EZ_CreateWidget(EZ_WIDGET_FRAME,NULL,
                          EZ_LABEL_STRING, "Drag target",
                          EZ_FILL_MODE, EZ_FILL_BOTH, 0);
  textW = EZ_CreateWidget(EZ_WIDGET_TEXT, frame, 0);

  EZ_WidgetAddDnDDataDecoder(textW, MY_FILE_NAME_ATOM, 0,
                             decodeFileName, NULL,
                             NULL, NULL);
  EZ_WidgetAddDnDDataDecoder(textW, MY_FILE_CONTENTS_ATOM, 0,
                             decodeFileContents, NULL,
                             NULL, NULL);
  EZ_DisplayWidget(frame);
  EZ_EventMainLoop();
}

int decodeFileName(EZ_Widget *widget, void *data,
                   char *message, int length)
{
  if(widget)
    {
      if(length > 0)
        {
          FILE *fp = fopen(message, "r");
          if(fp)
            {
              EZ_TextLoadFile(widget, message);
              fclose(fp);
              return(EZ_DND_SUCCESS);
            }
        }
    }
  return(EZ_DND_FAILURE);
}


int decodeFileContents(EZ_Widget *widget, void *data,
                       char *message, int length)
{
  if(widget)
    {
      if(length > 0)
        {
          EZ_FreezeWidget(widget);
          EZ_TextClear(widget);
          EZ_TextInsertString(widget, message);
          EZ_UnFreezeWidget(widget);
          return(EZ_DND_SUCCESS);
        }
```

```
    }
  return(EZ_DND_FAILURE);
}
```

Chapter 9

Messages

EZwgl includes a mechanism for clients to exchange messages. An *EZwgl* client can send a message to one or to all *EZwgl* clients. A message is a piece of text together with some control information encoded in the following data structure. The various fields are filled by the parties when messages are sent or received.

```
typedef struct {
  Atom       messageType;  /* message type identifier      */
  int        messageLength;/* the length of the message    */
  char       *message;     /* the message: textual string  */
  int        messageId;    /* set by sender, message Id    */
  int        replyId;      /* set by sender, reply id      */
  int        isReply;      /* True if the message is a reply  */
  /* src info */
  Window     sender;       /* sender's communication window id  */
  Atom       senderClass;  /* sender's class name: XrmQuark     */
  Atom       senderInstance; /* sender's instance name XrmQuark */
  int        senderIsSelf;   /* True if msg is originated from the client itself */
} EZ_Message;
```

Messages are handled by message handlers. A message handler is a procedure of the following type

`typedef  void (*EZ_MessageHandler)(EZ_Message *msgIn, void *client_data);`

To handle message, an application must register at least one message handler. Message handlers are registered by one of the following two commands.

```
void  EZ_RegisterMessageHandler(Atom mtype, int mId,  void *data,
                                EZ_MessageHandler handler, int timeout,
                                EZ_CallBack *timeoutCb, void *cdata);
void  EZ_SetDefaultMessageHandler(EZ_MessageHandler handler, void *data);
```

The first procedure registers a specific message handler. This handler will only handle messages of the specified type `mtype` and the specified message Id `mId`, and it will expire in `timeout` seconds if `timeout>0`. When it expires, it executes `timeroutCb(data, cdata)` if supplied. The second function registers the default message handler. The default message handler handles all messages which are not handled by specific handlers.

To delete a message handler, use

```
void  EZ_DeleteMessageHandler(Atom mtype, int mId,  void *data,
                              EZ_MessageHandler handler);
```

To send a message to a spcific client, use

```
void  EZ_SendMessage(Atom mtype, char *message, int length, int msgId,
                     Window recipient, int replayId, int isReply);
```

This function requires the id of the recipient's communication window, which is known only after some dialogue has been initiated. Normally, one broadcasts a message first to find an interested party. When the party replies, the id of its communication window is set in the `sender` field of the message.

The meaning of the arguments are:

- `mtype` specifies the message type.
- `message` specifies the message.
- `length` specifies the length of the message.
- `msgId` specifies the transaction serial number.

- `recipient` specifies the recipient.
- `replyId` if the sender requires a reply, it specifies the message id for the reply. If the recipient of this message decides to reply, it should use this message id..
- `isReply` specifies whether this message is a reply.

To broadcast a message to all *EZwgl* clients, use

```
void  EZ_BroadcastMessage(Atom mtype, char *message, int length,
                          int msgId, int replyId);
```

To reply a message, use

```
void  EZ_ReplyMessage  (EZ_Message *fromsg, char *reply, int replylength);
void  EZ_ReplyMessageR (EZ_Message *fromsg, char *reply, int len, int replyId);
```

The second function replies a message and requests a further correspondence from the recipient of the message.

9.1 An Example

In this section, we implement a simple client/server pair to demonstrate the use of message passing. The server will answer requests about creating pixmaps from image files. For demonstration purpose, it displays the messages together with the pixmaps in a text window. The client consists of a directory list tree and a label. When the user double click on a file item in the list tree, it broadcasts a message out asking the server to create a pixmap from the file.

The Server

```
#include "EZ.h"

Atom  PIXMAP_ATOM;

void msgHandler(EZ_Message *msg, void *data)
{
  /*
   * the default message handler for the server:
   * if message type is PXIMAP_ATOM, try to create
   * a pixmap from the file supplied, and send the
   * pixmap id and shape id back.
   */
  EZ_TextProperty         *tmpTp;
  char                    buf[1024], *ptr;
  EZ_Widget               *text = (EZ_Widget *)data;

  if(msg->messageType == PIXMAP_ATOM && msg->messageLength > 0)
    {
      char   reply[1024];
      Pixmap pixmap, shape;
      int    w, h;
      EZ_Pixmap *pptr;
      /* print the message */
      sprintf(buf, "\n+++From [%lx], Type=[%lx], Len=[%d] Id=[%d]+++\n   %s\n",
              msg->sender, msg->messageType, msg->messageLength,
              msg->replyId,  msg->message);
      EZ_TextInsertString(text, buf);

      /* allocate the pixmap */
      if((pptr = EZ_CreateLabelPixmapFromXpmFile(msg->message)) == NULL)
        pptr = EZ_CreateLabelPixmapFromImageFile(msg->message);
      if(pptr)
        {
          EZ_GetLabelPixmapInfo(pptr, &pixmap, &shape, &w, &h);
          /* reply */
          sprintf(reply, "%d %d %lx %lx", w, h, pixmap, shape);
          EZ_ReplyMessage(msg, reply, strlen(reply));

          /* display the pixmap */
          sprintf(buf, "   [pixmap=%lx shape=%lx]: replied\n", pixmap,shape);
          EZ_TextInsertString(text, buf);
          tmpTp =  EZ_GetTextProperty(EZ_LABEL_PIXMAP, pptr, NULL);
          EZ_TextInsertStringWithProperty(text, buf, tmpTp);
        }
      else
        {
          sprintf(buf, "  Couldn't create pixmap\n", pixmap,shape);
          EZ_TextInsertString(text, buf);
        }
    }
  else
    {
```

```
      sprintf(buf, "\n---From [%lx], Type=[%lx], Len=[%d] Id=[%d]----\n   %s\n",
              msg->sender, msg->messageType, msg->messageLength,
              msg->replyId,  msg->message);
      EZ_TextInsertString(text, buf);
    }
}

main(int ac, char **av)
{
  EZ_Widget *frame, *text;

  EZ_Initialize(ac, av, 0);
  PIXMAP_ATOM =  EZ_GetAtom("PixmapTestAtom");

  frame  = EZ_CreateWidgetXrm(EZ_WIDGET_FRAME,    NULL,
                              "server",           "Server",
                              EZ_FILL_MODE,       EZ_FILL_BOTH,
                              EZ_LABEL_STRING,    "Pixmap server",
                              EZ_IPADX,           4,
                              EZ_SIZE,            400, 400,
                              0);
  text =  EZ_CreateWidgetXrm(EZ_WIDGET_TEXT,      frame,
                             "text",              "Text",
                              0);
  EZ_SetDefaultMessageHandler(msgHandler, text);

  EZ_DisplayWidget(frame);
  EZ_EventMainLoop();
}
```

The Client

```
#include "EZ.h"

Atom  PIXMAP_ATOM;

void msgHandler(EZ_Message *msg, void *data)
{
  EZ_Widget        *label = (EZ_Widget *)data;
  if(msg->messageLength > 0)
    {
      Pixmap pixmap, shape;
      int    w, h;
      if(sscanf(msg->message, "%d %d %lx %lx", &w, &h, &pixmap, &shape) == 4 && w > 0)
        {
          EZ_Pixmap *ptr = EZ_CreateLabelPixmapFromXPixmap(pixmap, shape, 0, 0, w,h, 1);
          if(ptr) EZ_ConfigureWidget((EZ_Widget *)data, EZ_LABEL_PIXMAP, ptr, 0);
          /* delete this handler */
          EZ_DeleteMessageHandler(msg->messageType, msg->messageId, data, msgHandler);
        }
    }
}

static void timeoutCb(void *notused, void *data)
{
  printf("message handler timeout\n");
}

void ltreeCB(EZ_Widget *tree, void *data)
{
  EZ_TreeNode  *node = EZ_GetListTreeWidgetSelection(tree);
  EZ_Item      *item;
  static       int  msgId = 1;

  if(node && (item = EZ_TreeNodeGetItem(node)))
    {
      char *path = EZ_GetDirTreeNodeFullPath(node);
      /* add a message handler */
      EZ_RegisterMessageHandler(PIXMAP_ATOM, /* type of msg it handles */
                                msgId+1,     /* the msgId it handles   */
                                data,        /* client data to be passed to handler */
                                msgHandler,  /* the handler            */
                                1,           /* expire in one second   */
                                timeoutCb,   /* what to do when handler expires */
                                data         /* client data to be passed to timeoutCb */
                                );
      /* broadcast a message to find a server */
      EZ_BroadcastMessage(PIXMAP_ATOM,  /* msg type */
                          path,         /* the message */
                          strlen(path), /* length of message */
                          msgId,        /* this message Id   */
                          msgId+1       /* ask recipient to reply using this id */
                          );
      msgId += 2;
    }
}
```

```
main(int ac, char **av)
{
  EZ_Widget *frame, *ltree, *label, *phandle;
  EZ_TreeNode *root;

  EZ_Initialize(ac, av, 0);
  PIXMAP_ATOM =  EZ_GetAtom("PixmapTestAtom");

  frame  = EZ_CreateWidgetXrm(EZ_WIDGET_FRAME,    NULL,
                              "server",           "Server",
                              EZ_FILL_MODE,       EZ_FILL_BOTH,
                              EZ_LABEL_STRING,    "Pixmap client",
                              EZ_IPADX,           4,
                              EZ_SIZE,            800, 400,
                              0);
  ltree =  EZ_CreateWidgetXrm(EZ_WIDGET_TREE,      frame,
                              "tree",              "Tree",
                              EZ_WIDTH,            300,
                              0);
  phandle = EZ_CreateWidget(EZ_WIDGET_PANE_HANDLE, frame,
                            0);

  label = EZ_CreateWidgetXrm(EZ_WIDGET_LABEL,      frame,
                             "label",              "Label",
                             0);
  EZ_AddWidgetCallBack(ltree, EZ_CALLBACK, ltreeCB, label, 0);

  EZ_GlobHiddenFiles(True);
  root = EZ_CreateDirTree(NULL, "./*", NULL, 0);  /* */
  EZ_SetListTreeWidgetTree(ltree, root);

  EZ_DisplayWidget(frame);
  EZ_EventMainLoop();
}
```

Chapter 10

Display Items

A *display item* is an object which contains some visual information and the rules to display them. Display items share the same set of visual characteristics as that of widgets, like fonts, borders, background and foreground colors. They also share some functional attributes as that of widgets, like callbacks and drag and drop. Display items also support resources (they are treated the same as toplevel widgets by the resource manager).

However, display items and widgets are fundamentally different entities. Widgets directly communicate with the user and the X server while display items must rely on their host widgets to fullfil the interaction.

Display items are used in *FancyListBox, ListTree* and *WorkArea* widgets.

The following types of display items are currently supported.

- `EZ_WIDGET_ITEM`
- `EZ_LABEL_ITEM`
- `EZ_FIG_ITEM`

10.1 Widget Item

Widget items are fully functional widgets. They are very different from other items. For a label item or a fig item, interactions between the user and the item are handled by the middle man, its host widget. For a widget item, however, user interactions are directly handled by the relevent widget. For this reason, callbacks registered to a widget item will probably never get called.

10.1.1 Resources

Widget item support the following resources.

configOption	resourceName	valueType
EZ_X	x	int
EZ_Y	y	int
EZ_LOCATION	location	int_x int_y
EZ_WIDTH	width	int
EZ_HEIGHT	height	int
EZ_SIZE	size	int_w int_h
EZ_GEOMETRY	geometry	int_x int_y int_w int_h

configOption	resourceName	valueType
EZ_PADX	padx	enum
EZ_PADY	pady	int
EZ_BORDER_TYPE	borderType	enum
EZ_BORDER_WIDTH	borderWidth	int
EZ_BACKGROUND	background	string

10.1.2 Interface Functions

```
void EZ_SetWidgetItemWidget(EZ_Item *item, EZ_Widget *widget);
```

set the widget of a widget item.

```
EZ_Widget  *EZ_GetWidgetItemWidget(EZ_Item *item);
```

return the widget of a widget item.

10.2 Label Item

A label item displays a piece of textual string or a pixmap/image or both, in the later case, it may be orientated either horizontally or vertically.

10.2.1 Resources

Label item supports the following resources.

configOption	resourceName	valueType
EZ_X	x	int
EZ_Y	y	int
EZ_LOCATION	location	int_x int_y
EZ_WIDTH	width	int
EZ_HEIGHT	height	int
EZ_SIZE	size	int_w int_h
EZ_GEOMETRY	geometry	int_x int_y int_w int_h

configOption	resourceName	valueType
EZ_PADX	padx	enum
EZ_PADY	pady	int
EZ_BORDER_TYPE	borderType	enum
EZ_BORDER_WIDTH	borderWidth	int
EZ_FOREGROUND	foreground	string
EZ_BACKGROUND	background	string
EZ_ORIENATTION	orientation	enum

configOption	resourceName	valueType
EZ_LABEL_STRING	labelString	string
EZ_FONT_NAME	fontName	string
EZ_FONT_ID	fontId	int
EZ_TEXT_LINE_LENGTH	textLineLength	int
EZ_JUSTIFICATION	justification	enum
EZ_LABEL_POSITION	labelPosition	enum
EZ_PIXMAP_FILE	pixmapFile	string
EZ_IMAGE_FILE	imageFile	string
EZ_BITMAP_DATA	N/A	string
EZ_PIXMAP_DATA	N/A	char **
EZ_X_PIXMAP	N/A	Pixmap
EZ_LABEL_PIXMAP	N/A	EZ_Bitmap *

configOption	resourceName	valueType
EZ_DND_BUBBLE_STRING	dndBubbleString	string
EZ_DND_DRAG_CURSOR	dndDragCursor	string

configOption	resourceName	valueType
EZ_CALLBACK	N/A	void* void*
EZ_DESTROY_CALLBACK	N/A	EZ_CallBack void*

10.2.2 Interface Functions

`void EZ_GetLabelItemStringInfo (EZ_Item *item, char **str, int *len);`

return the label string of a label item.

`int EZ_GetLabelItemTextGeometry(EZ_Item *item, int *x, int *y, int *w, int *h);`

return position and text extent of the label string in a label item.

`EZ_Bitmap   *EZ_GetLabelItemPixmap (EZ_Item *item);`

return the label pixmap of a label item.

`void EZ_LabelItemSetProperty(EZ_Item *item, EZ_TextProperty *prop);`

set or modify the visual info of a label item via text property.

10.3 Figure Item

A figure item is a free style display item. One can draw lines, polygons, arcs, text and pixmaps in figure item. Coordinates in a figure item are relative to its display location and a measured in pixels.

10.3.1 Resources

Figure item supports the following resources.

configOption	resourceName	valueType
EZ_X	x	int
EZ_Y	y	int
EZ_LOCATION	location	int_x int_y
EZ_WIDTH	width	int
EZ_HEIGHT	height	int
EZ_SIZE	size	int_w int_h
EZ_GEOMETRY	geometry	int_x int_y int_w int_h

configOption	resourceName	valueType
EZ_BORDER_TYPE	borderType	enum
EZ_BORDER_WIDTH	borderWidth	int
EZ_BACKGROUND	background	string

10.3.2 Interface Functions

```
GC  EZ_GetGC(unsigned long mask, XGCValues *gcv);
```

find or create a GC that matches the specified characteristics.

```
void EZ_FreeGC(GC gc);
```

free a GC allocated by `EZ_GetGC`.

The following routines are wrappers for the corresponding Xlib drawing procedures.

```
EZ_FigPiece  *EZ_FigItemAddString(EZ_Item *fitem, GC theGC,
                       char *str, int slen, int justification, int x, int y);
```

add a string to a figure item.

```
EZ_FigPiece  *EZ_FigItemAddPixmap(EZ_Item *fitem, GC theGC,
              EZ_Bitmap *bmp, int sx, int sy, int sw, int sh, int dx, int dy);
```

add a pixmap to a figure item.

```
EZ_FigPiece  *EZ_FigItemAddPoints(EZ_Item *fitem, GC theGC,
                                  XPoint *pts, int npts);
```

add some points to a figure item.

```
EZ_FigPiece  *EZ_FigItemAddLines (EZ_Item *fitem, GC theGC,
                          XPoint *pts, int npts, int mode);
```

add some lines to a figure item.

```
EZ_FigPiece  *EZ_FigItemAddPolygon(EZ_Item *fitem, GC theGC,
                           XPoint *pts, int npts, int mode);
```

add a polygon to a figure item.

```
EZ_FigPiece  *EZ_FigItemAddRectangle(EZ_Item *fitem, GC theGC,
              int x, int y, int w, int h);
EZ_FigPiece  *EZ_FigItemAddFilledRectangle(EZ_Item *fitem, GC theGC,
              int x, int y, int w, int h);
EZ_FigPiece  *EZ_FigItemAddRectangles(EZ_Item *fitem, GC theGC,
               XRectangle *rects, int nrects);
EZ_FigPiece  *EZ_FigItemAddFilledRectangles(EZ_Item *fitem, GC theGC,
              XRectangle *rects, int nrects);
```

add a rectangles/filled rectangles to a figure item.

```
EZ_FigPiece  *EZ_FigItemAddArc(EZ_Item *fitem, GC theGC,
              int x, int y, int w, int h, int angle1, int angle);
EZ_FigPiece  *EZ_FigItemAddFilledArc(EZ_Item *fitem, GC theGC,
               int x, int y, int w, int h, int a1, int a2);
EZ_FigPiece  *EZ_FigItemAddArcs(EZ_Item *fitem, GC theGC,
               XArc *arcs, int narcs);
EZ_FigPiece  *EZ_FigItemAddFilledArcs(EZ_Item *fitem, GC theGC,
               XArc *arcs, int narcs);
```

add an arc/filled arc to a figure item

```
EZ_FigPiece  *EZ_FigItemAddSegments(EZ_Item *fitem, GC theGC,
              XSegment *segs, int nsegs);
```

add line segments to a figure item.

The following are figure item/piece manipulation commands.

```
void EZ_ScaleFigItem(EZ_Item *item, float sx, float sy, int mode);
```

scale a figure item.

```
void EZ_FigItemDeletePiece(EZ_Item *item, EZ_FigPiece *fpiece, int update);
```

delete a figure piece from a figure item.

```
void EZ_FigItemHidePiece(EZ_Item *item, EZ_FigPiece *fpiece, int update);
```

hide a figure piece from a figure item.

```
void EZ_FigItemShowPiece(EZ_Item *item, EZ_FigPiece *fpiece, int update);
```

un-hide a figure piece.

```
void EZ_GetFigPieceBoundingBox(EZ_FigPiece *fpiece,
        int *x1, int *y1, int *x2, int *y2);
```

return the bounding box of a figure piece.

10.4 Write Your Own Display Items

EZwgl also provides a way for you to write your own display items and make it available to `EZ_CreateItemXrm`. To do so, you need to

- provide the following 8 procedures

```
setup(EZ_Item *item);
computeSize(EZ_Item *item, int *w, int *h);
draw(EZ_Widget *widget, Drawable  drawable, int x0, int y0,
     EZ_Item   *item, int Rx, int Ry, int Rw, int Rh);
update(EZ_Widget *widget, Drawable  drawable,
       EZ_Item   *item, int Rx, int Ry, int Rw, int Rh);
highlight(EZ_Widget  *widget, Drawable drawable,
          EZ_Item  *item, unsigned long fg, unsigned long bg,
          int Rx, int Ry, int Rw, int Rh, int flag);
freePrivateData(EZ_Item *item);
int pick(EZ_Item *item, int x, int y);
postscript (EZ_Item *item, FILE *fp, void *psinfo);
```

- call `EZ_RegisterItem` to register your item. This function returns an integer type identifier which can be passed to `EZ_CreateItem` to instantiate the registered item.
- write interface routines as needed.

—details to come—

10.5 Commands Common to All Display Items

Here is a list of commands common to all display items.

```
EZ_Item *EZ_DuplicateItem (EZ_Item *item);
```

duplicate a figure item or a label item (not available for widget item).

```
void EZ_DestroyItem(EZ_Item *item);
```

delete a display item.

```
int EZ_GetItemType(EZ_Item *item);
```

return the type of a display item.

```
void EZ_GetItemBwAndXYPading(EZ_Item *item, int *bw, int *xpad, int *ypad);
```
return border width and paddings of a display item.

```
void  EZ_GetItemClientData(EZ_Item *item, int *i, void **p);
```
return the client data attached to a display item.

```
int EZ_GetItemIntData(EZ_Item *item);
```
return the integer attached to a display item.

```
void *EZ_GetItemPtrData(EZ_Item *item);
```
return the pointer attached to a display item.

```
void  EZ_SetItemClientData(EZ_Item *item, int i, void *p);
```
attach client data to a display item.

```
void EZ_SetItemIntData(EZ_Item *item, int i);
```
attach an integer to a display item.

```
void EZ_SetItemPtrData(EZ_Item *item, void *p);
```
attach a pointer to a display item.

```
void EZ_AddItemCallBack(EZ_Item *item, int which,
        EZ_CallBack callback, void *cdata, int where);
```
add a callback or a destroy callback to a display item.

```
void EZ_RemoveItemCallBack(EZ_Item *item,  int which,
       EZ_CallBack callback, void *cdata);
```
remove a callback/destroy callback from a display item.

```
void EZ_RemoveAllItemCallBacks(EZ_Item *item, int which);
```
remove all callbacks/destroy callbacks.

```
EZ_Widget *EZ_GetItemHostWidget(EZ_Item *item);
```
return the host widget of a display item.

```
void EZ_UpdateDisplayItem (EZ_Item *item);
```
redisplay a display item.

```
void EZ_SetItemPosition(EZ_Item *item, int x, int y);
```
set the location of a display item.

```
void EZ_GetItemPosition(EZ_Item *item, int *x, int *y);
```
return the configured/computed location of a display item.

```
void EZ_GetItemAPosition(EZ_Item *item, int *x, int *y);
```
return the actual/displayed location of a display item.

```
void EZ_SetItemDimension(EZ_Item *item, int w, int h);
```
set the dimension of a display item.

```
void EZ_GetItemDimension(EZ_Item *item, int *w, int *h);
```
return the configured/computed dimension of a display item.

```
void EZ_GetItemADimension(EZ_Item *item, int *w, int *h);
```
return the actual/displayed dimension of a display item.

```
int EZ_PickAnItem( EZ_Item *item, int x, int y);
```
return True if point (x,y) is on item.

Chapter 11

Trees and Direcory Trees

A *tree* is a special arrangement of display items. A directory tree is a graphical representation of the unix file system. A tree may be displayed in three styles: tree, list tree or branched list tree. Tree displaying is controled by three parameters, indent, line spacng and draw-branch. These attributes is set by the command `EZ_TreeSetCharacter`.

11.1 Trees

Trees are built from the special data structure `EZ_TreeNode`. A tree node is created by the procedure

```
EZ_TreeNode *EZ_CreateTreeNode(EZ_TreeNode *parent, EZ_Item *item);
```

The *tree* object is used only in the `Tree` widget.

EZwgl includes commands for manipulating trees as well as special commands for manipulating directory trees.

11.1.1 Interface Routines

```
EZ_TreeNode  *EZ_TreeNodeGetParent(EZ_TreeNode *node);
```

return the parent node.

```
EZ_TreeNode  *EZ_TreeNodeGetChildren(EZ_TreeNode *node);
```

return the first child.

```
EZ_TreeNode  *EZ_TreeNodeGetSibling(EZ_TreeNode *node);
```

return the sibling node.

```
EZ_TreeNode  *EZ_TreeFindNode(EZ_TreeNode *root, EZ_TreeNode *node);
```

find a node in a tree.

```
EZ_TreeNode  *EZ_TreeFindNodeUsingItem(EZ_TreeNode *root, EZ_Item *item);
```

find a node using display item as the search key.

```
EZ_TreeNode  *EZ_TreeFindNodeUsingProcedure(EZ_TreeNode *root,
 EZ_TreeNode *(*f)(EZ_TreeNode *node, void *data), void *cdata );
```

search for a tree node using the specified procedure.

```
void EZ_TreeDestroySubtree(EZ_TreeNode *node);
```

destruct a subtree.

```
void EZ_TreeDestroyNodeDescendants(EZ_TreeNode *node);
```

destroy all descendants of a tree node.

```
void EZ_TreeAttachSubtree(EZ_TreeNode *node, EZ_TreeNode *subtree);
```

append a subtree to a tree node.

```
void  EZ_TreeDetachSubtree(EZ_TreeNode *node);
```

detach a node and its descendants from a tree.

```
void EZ_TreeSetCharacter(EZ_TreeNode *node,
 int type, int indent, int spacing, int drawBranch);
```

set the display characteristics for a tree.

```
void EZ_TreeTraverseTree(EZ_TreeNode *node,
        void (*preF)(EZ_TreeNode *node)),
        void (*postF)(EZ_TreeNode *node) );
```

traverse a tree and execute the supplied procedures on each node.

```
void EZ_TreeSetFGColor(EZ_TreeNode *node, unsigned long pv);
void EZ_TreeSetFGColorName (EZ_TreeNode *node, char *name);
```

set the foreground color for all nodes.

```
void EZ_TreeSetBGColor(EZ_TreeNode *node, unsigned long pv);
void EZ_TreeSetBGColorName(EZ_TreeNode *node, char *name);
```

set the background color for all nodes.

```
void EZ_ComputeSubtreeSize(EZ_TreeNode *node, int *w, int *h, int *parity);
```

compute the minimal rectangle needed to hold a subtree.

```
void EZ_DisplayTreeInDrawable(EZ_Widget *widget, Drawable drawable,
          int x0, int y0, EZ_TreeNode *node,
          int Rx, int Ry, int Rw, int Rh);
```

display a subtree.

```
EZ_Item *EZ_TreeNodeGetItem(EZ_TreeNode *node);
```

return the item assigned to a tree node.

11.2 Directory Trees

EZwgl provides a few convenience routines for programmers to manipulating directory trees.

A directory tree is created by calling

```
EZ_TreeNode *EZ_CreateDirTree(EZ_DirTreeNodeCreator creator,
            char *patterns, void *client_data, int open_mode);
```

where `creator` is a function of type

```
 EZ_TreeNode *creator(char *path, void *client_data);
```

`pattern` is a glob pattern such as `"/home/mzou/*"` or `"*.c *.h"`;
`client_data` is an arbitary client data to be passed to `creator` and
`open_mode` is a flag which specifies how to open a directory node when a double button1 clicks occurs on the node. Must be

```
EZ_DIR_NODE_OPEN_DIRECTORY_NONE
EZ_DIR_NODE_OPEN_DIRECTORY_DEFAULT
EZ_DIR_NODE_OPEN_DIRECTORY_COLLAPSING
```

11.2.1 Interface Routines

```
void EZ_OpenCloseDirTreeDirectoryNode(EZ_Item *item, int collaps);
```

open or close (depends on the current status) a directory node.

```
char *EZ_GetDirTreeNodeFullPath(EZ_TreeNode *node);
```

return the full pathname of a dir tree node.

```
void EZ_SetDirTreeOpenDirectoryMode(EZ_TreeNode *node, int mode);
```

set the openning mode of a dir tree.

```
void EZ_SetDirTreeDefaultPixmaps(EZ_Bitmap *o, EZ_Bitmap *c, EZ_Bitmap *f);
void EZ_SetDirTreeDefaultPixmapsFromXpmFiles(char *o, char *c, char *f);
```

set the three default label pixmaps for directory trees.

```
void  EZ_UpdateDirTreeWidget(EZ_Widget *TreeWidget);
```

update a dir tree widget.

Chapter 12

Miscellaneous

12.1 About Pixmap Labels

The *EZwgl* library keeps a hash table for pixmaps used in an application. Each entry in the hash table has a reference count. When the reference count reaches 0, the entry is de-allocated and the relevent X resources are released. If an application uses lots of pixmaps and needs to switch constantly between different pixmaps, chances are that these pixmaps are allocated and de-allocated frequently. As a result, the performance may not be optimal since allocating pixmaps, especially creating pixmaps from images are expensive. For this situation, one may explicitly allocate the pixmaps first and then use the internal pixmaps. Explicitly allocated pixmaps are guaranteed to be cached until being explicitly freed.

The following code segment demonstrates this usage.

```
EZ_LabelPixmap *pixmap1 = EZ_CreateLabelPixmapFromImageFile(``file1'');
EZ_LabelPixmap *pixmap2 = EZ_CreateLabelPixmapFromImageFile(``file2'');

...

if(condition1) /* use the first pixmap */
  EZ_ConfigureWidget(widget, EZ_LABEL_PIXMAP, pixmap1, 0);
else /* use the second pixmap */
  EZ_ConfigureWidget(widget, EZ_LABEL_PIXMAP, pixmap2, 0);

...
```

There are five rountines in *EZwgl* that allow an application to create internal pixmaps from different sources.

```
EZ_LabelPixmap *EZ_CreateLabelPixmapFromXpmFile(char *fileName);
EZ_LabelPixmap *EZ_CreateLabelPixmapFromImageFile(char *fileName);
EZ_LabelPixmap *EZ_CreateLabelPixmapFromXBitmapFile(char *fileName);
EZ_LabelPixmap *EZ_CreateLabelPixmapFromXpmData(char **data);
EZ_LabelPixmap *EZ_CreateLabelPixmapFromXBitmapData(char *data,
                                     int width, int height);
EZ_LabelPixmap *EZ_CreateLabelPixmapFromXPixmap(Pixmap pixmap, Pixmap shape,
                                     int x, int y, int width, int height, int reuse);
```

To free an internal label pixmap, use

```
void EZ_FreeLabelPixmap(EZ_LabelPixmap *IPixmap);
```

Another situation is when an application needs to use part of an image to label a widget. *EZwgl* provides an indirect way to achieve this goal. One first creates a native X pixmap by using `EZ_CreateXPixmapFrom***`, or `XCreatePixmap`, then uses the `EZ_X_PIXMAP` configuration option to set the label.

The following code segement is an example.

```
Pixmap pixmap;
int    width, height;

if(EZ_CreateXPixmapFromImageFile(``file'',&width, &height, &pixmap))
  {
     /* use the upper left quarter of the image to label widget */
     EZ_ConfigureWidget(widget,
                        EZ_X_PIXMAP, pixmap, 0, 0, width/2, height/2,
```

```
                    0);

      /* if there are no other use of pixmap, free it */
      EZ_FreeXPixmap(pixmap);
    }

  ...
```

Besides the Xlib function `XCreatePixmap`, the following five functions may also be used to create native pixmaps from various sources.

```
int EZ_CreateXPixmapFromXpmFile(char *fileName, int *width_ret,
        int *height_ret, Pixmap *pixmap_ret, Pixmap *shape_ret);
int EZ_CreateXPixmapFromImageFile(char *fileName,
        int *width_ret, int *height_ret, Pixmap *pixmap_ret);
int EZ_CreateXPixmapFromBitmapFile(char *fileName, int *width_ret,
        int *height_ret, Pixmap *pixmap_ret, Pixmap *shape_ret);
int EZ_CreateXPixmapFromXpmData(char **data, int *width_ret,
        int *height_ret, Pixmap *pixmap_ret, Pixmap *shape_ret);
```

Native X pixmaps created by `EZ_CreateXPixmapFrom***` are retained until one call one of the following.

```
void XFreePixmap(EZ_GetDisplay(), pixmap);
void EZ_FreeXPixmap(Pixmap pixmap);
```

The next routine returns the information about a LabelPixmap.

```
void EZ_GetLabelPixmapInfo(EZ_Pixmap *pix, Pixmap *pix_ret,
             Pixmap *shape_ret, int w_ret, int h_ret);
```

Chapter 13

Using the Graphics Library

The EZ graphics library consists about 100 or so OpenGL^{TM} like routines that provides a fairly complete set of graphics primitives for doing 3D graphics programming under the X window system.

There are two important data types used in the EZ graphics library.

```
typedef  float          EZ_Vector[4];
typedef  float          EZ_Matrix[4][4];
```

Internally, all geometric primitives are described using `EZ_Vector` and are transformed using `EZ_Matrix`.

13.1 Getting Started

Roughly speaking, using the EZ graphics library consists of 6 steps.

- Initialize *EZwgl*, Create a 3DCanvas and display it. If needed, an application may create more than one 3DCanvases. However, only one of them is active at any given time. Use `EZ_Set3DCanvas` to set the active 3DCanvas.
- Set the display attributes. The important ones are *color mode* and *back buffer type* (if double buffer is used). Color mode is sort of global across all 3DCanvases since the graphics library uses a single physical colormap. While other attributes, like back buffer, zbuffer, lighting, color material, back face culling, foging etc are binded to individual 3DCanvases.
- Initialize GL data. This include matrices, lighting parameters, material properties etc.
- Create and display geometric objects by using `EZ_Begin`, `EZ_End`, `EZ_Vertex3f{v}`, `EZ_Normal3f{v}`, `EZ_Color3f{v}`, `EZ_Lightf{v}` etc. You may also use display list.
- Create a event handler routine for each 3DCanvas, this part is the trickest.
- Processing events by using any one of the three event handling routines.

13.1.1 Example G1, A Polygon

Let's start with a small example. We'll create a 3DCanvas widget, display it and draw a shaded polygon in it. It demonstrate the steps outlined above.

```
/*************** Example G1 ******************************************/
#include "EZ.h"

static void eventHandler(EZ_Widget *, void *, int, XEvent *);
static void draw(EZ_Widget *);

main(int argc, char **argv)
{
  EZ_Widget *frame, *canvas;
  /*
   * Initialize EZWGL, don't forget to
   * initialize the graphics library also.
   */
  EZ_Initialize(argc, argv, 1);      /* notice the 1        */
  /*
   * canvases do not resize propertly as toplevel
   * widgets (bug!) It is recommended to put it
   * under a frame at least for now.
   */
  frame = EZ_CreateWidget(EZ_WIDGET_FRAME, NULL,
```

```
                         EZ_FILL_MODE, EZ_FILL_BOTH,
                         0);
  canvas = EZ_CreateWidget(EZ_WIDGET_3D_CANVAS, frame,
                           EZ_WIDTH_HINT,  320,
                           EZ_HEIGHT_HINT, 320,
                           EZ_EVENT_HANDLE, eventHandler, NULL,
                           0);
  /*
   * now display the canvas. One must first display
   * a 3DCanvas before calling any GL functions !!!
   */
  EZ_DisplayWidget(canvas);

  /* now setup global GL attributes */
  EZ_RGBMode();                 /* we are using RGB mode  */
  EZ_ShadeModel(EZ_SMOOTH);     /* turn on smooth shading */
  /*
   * We'll draw a 2D polygon, so we don't have to setup
   * a complex projection matrix. The default matrix mode
   * is EZ_MODELVIEW. We just set the projection matrix
   * on this default matrix stack.
   */
  EZ_LoadIdentity();            /* clear the top mtx 2 id */
  EZ_Ortho2(-1.0,1.0,-1.0,1.0); /* project the unit cube  */

  draw(canvas);                 /* draw the polygon       */
  EZ_EventMainLoop();           /* handle events          */
}

static void draw(EZ_Widget *canvas)
{
  EZ_Clear(EZ_COLOR_BUFFER_BIT); /* clear the frame buffer */

  EZ_Begin(EZ_POLYGON);          /* start a polygon       */
  EZ_Color3f(1.0,0.0,0.0);       /* each vertex has its clr*/
  EZ_Vertex2f(1.0,0.0);
  EZ_Color3f(1.0,1.0,0.0);
  EZ_Vertex2f(0.5,0.866);
  EZ_Color3f(0.0,1.0,0.0);
  EZ_Vertex2f(-0.5,0.866);
  EZ_Color3f(0.0,1.0,1.0);
  EZ_Vertex2f(-1.0,0.0);
  EZ_Color3f(0.0,0.0,1.0);
  EZ_Vertex2f(-0.5,-0.866);
  EZ_Color3f(1.0,0.0,1.0);
  EZ_Vertex2f(0.5,-0.866);
  EZ_End();                     /* the end of the polygon */
}
/*
 * the event handler. It should handle at least the
 * EZ_REDRAW event and the EZ_RESIZE.
 */
static void eventHandler(EZ_Widget *canvas, void *data,
                         int eventType, XEvent *xevent)
{
  switch(eventType)
    {
    case EZ_REDRAW:
    case EZ_RESIZE:
      draw(canvas);
      break;
    case EZ_KEY_PRESS:
      if(EZ_PressedKey == EZ_ESCAPE_KEY) exit(0);
      break;
    default:
      break;
    }
}
/*************** Example G1 *****************************************/
```

13.2 The 3DCanvas Widget

The $\mathcal{EZ}$ graphics library is realized through the 3DCanvas widget. To each 3DCanvas widget, there is an associated data structure called *the graphics context* which contains the entire state attributes of the graphics library. An application can have as many 3DCanvases as it needs. However, there is only one *active 3DCanvas* at any given instant. Implicitly, the active 3DCanvas is set to the last created 3DCanvas.

Start with release 1.38, borders for 3DCanvas widget has been removed. Supporting of border causes various unpleasent problems.

To use the graphics library, one must first create some 3DCanvases and display them. The graphics library comes to existence only after an active 3DCanvas has been displayed.

If more than one 3DCanvases are needed. Use `EZ_Set3DCanvas` to set the active 3DCanvas. This function performs a graphics context switch.

Events for 3DCanvas are handled exactly the same as for the other widget in *EZwgl*. The difference is that 3DCanvas does not have a default event handler while other widgets do.

One should register at least one private event handler for each 3DCanvas an application creates. The private event handler should provide means to handle at least the `EZ_REDRAW` and the `EZ_RESIZE` events.

Event handlers are registered via `EZ_AddEventHander` or by

Here is an example of event handlers.

```
void SampleEventHandler(EZ_Widget *canvas, void *vd, int et, XEvent *xev)
{
  switch(et) {
       case EZ_REDRAW:
       case EZ_RESIZE:
         /* redraw objects in canvas */
         break;
       case EZ_KEY_PRESS:
         /* do something  */
         break;
       default:
         break;
   }
}
```

EZwgl provides three additional commands for an application manipulate events for 3DCanvas widgets.

`void EZ_GLResetEvents(EZ_Widget *canvas)`

This function clears the event queue for the `canvas`. If `canvas` is `NULL`, it clears the event queue for the current active 3DCanvas widget.

`void EZ_GLEnterEvent(EZ_Widget *canvas, int eType, int *values)`

This function enters the specified event to `canvas` or the current active 3DCanvas if `canvas` is `NULL`. `eType` is one of

```
EZ_REDRAW
EZ_RESIZE
EZ_KEY_PRESS
EZ_BUTTON1_PRESS               EZ_BUTTON1_RELEASE
EZ_BUTTON2_PRESS               EZ_BUTTON2_RELEASE
EZ_BUTTON3_PRESS               EZ_BUTTON3_RELEASE
EZ_ENTER_WINDOW                EZ_LEAVE_WINDOW
EZ_POINTER_MOTION
```

For events of type `EZ_BUTTON?_PRESS`, `EZ_BUTTON?_RELEASE` or `EZ_POINTER_MOTION`, `values`, if not `NULL` is an array of two integers which specifies the pointer location in the current viewport. (Origin is at the lower-left corner!) For `EZ_KEY_PRESS` event, `values`, if not `NULL` is a pointer to an integer which specifies the keycode for the key that is being pressed.

The window coordinate system for a 3DCanvas is different from that of other widgets. For a 3DCanvas, the origin is the lower-left corner of the widget window while for other widgets, the origin is the upper-lefte corner of the widget window.

`int EZ_GLNextEvent(EZ_Widget *canvas, XEvent *xevent)`

This function checks Xlib's event queue for any outstanding events. If the event queue is empty, it waits for the next event to arrive. It then removes the first event on the queue and copy it to `event` if `event` is not `NULL`. If the removed event belongs to `canvas`, it returns the corresponding type. If the event does belong to `canvas`, it dispatches the event to the relevent widget and returns `0`.

The *EZ* graphics library also provides a user convenience routine to save the contents of a 3DCanvas window.

```
void EZ_Save3DCanvas2PPMImage(EZ_Widget *canvas, chae *file_name);
void EZ_Save3DCanvas2PS(EZ_Widget *canvas, char *fileName);
void EZ_Save3DCanvas2PSA(EZ_Widget *canvas, int mode, int dpi,
                float scale, char *fileName);
```

13.3 Pre-allocated Colors

The pre-allocated color entries are listed in the following table.

index	color name	RGB triple
BLACK (0)	black	(0,0,0)
GRAY1 (1)	gray38	(97,97,97)
GRAY2 (2)	gray46	(117,117,117)
GRAY3 (3)	gray50	(150,150,150)
GRAY4 (4)	gray74	(189,189,189)
GRAY5 (5)	gray84	(214, 214, 214)
GRAY6 (6)	gray87	(222,222,222)
GRAY7 (7)	gray93	(237,237,237)
WHITE (8)	gray100	(255,255,255)
DARKRED (9)	red4	(139,0,0)
RED (10)	red1	(255,0,0)
GREEN (11)	green1	(0,255,0)
BLUE (12)	blue1	(0,0,255)
YELLOW (13)	yellow	(255,255,0)
MAGENTA (14)	magenta	(255,0,255)
CYAN (15)	cyan	(0,255,255)

Entries 1–7 are used for displying widget borders. If a global border has been set, or an application uses the `-background` command line option, these entries may be modified.

13.4 Setting the Display Mode

Most of the graphics state attributes can be set via the command `EZ_Enable` (see next section). However, there are two important attributes that must be set via separate function calls. These two attributes are *ColorMode* and *BackBufferType*.

13.4.1 Choosing a Color Mode

The *EZ* graphics library supports both RGB mode and Colormap (index) mode. The color mode is binded to 3DCanvases, i.e., each 3DCanvas may choose either of the two color modes. But realistically, the two color modes cannot be used simultaneously on PseudoColor displays since *EZwgl* uses a single physical colormap.

```
void EZ_RGBMode(void);
```

This function sets the display mode to RGB mode. On 8bit displays, *EZwgl* uses dithering to simulate a 24 bit RGB Mode.

In RGB mode, colors are referenced directly from their RGB intensities. There are two pairs of functions for setting the current color in RGB mode.

```
void EZ_Color3f(float r, float g, float b)
void EZ_Color3fv(float rgb[3])
```

```
void EZ_Color3i(int r, int g, int b)
void EZ_Color3iv(int rgb[3])
```

All these routines set the current RGB color. In the '`f`' version, RGB intensities are in the range [0.0, 1.0] while in the '`i`' version, intensities are in the range [0, 255].

```
void EZ_ColorMapMode(void)
```

This function sets the display mode to index mode. Care must be taken in index mode since we only have a very limited number (less than 256) of colors.

In ColorMap mode, colors are refered through a lookup table (internal color map). Entries of this table can be set/modified by calling the function

EZwgl keeps a global shared colormap of size 256. This colormap is shared by all widgets. On PseudoColor/GrayScale displays, it is not even guaranteed that one can allocate all 256 colors.

```
void EZ_MapColor(int idx, int r, int g, int b);
```

To set the current color in ColorMap mode, use

```
void EZ_Index(int idx);
void EZ_Indexf(float fidx)
```

13.4.2 Choosing a Back Buffer

Be default, the library is in single buffer mode. Whatever you draw is displayed immediately on the screen. If you want to do some animations, single buffer will not be adequate because you'll see the screen clear as well as all the drawing. During animation, you'll see a visible flickering. For such situation, it is preferable to draw a frame in a hidden back buffer and copy the completed buffer to the display (the front buffer) when done.

Buffering is controled by three items: A back buffer, an attribute to signal which buffer to draw into, and a command to copy the back buffer to the front buffer.

The following routines select a back buffer.

`void EZ_SetBackBuffer(int type);`

This function chooses a back buffer for the current graphics context. *EZwgl* supports two types of back buffers, an `EZ_PIXMAP` back buffer and an `EZ_XIMAGE` back buffer. The latter is usually faster on localhost. On a remote host, however, the former is usually faster since it needs less data transfer via the network.

The `EZ_XIMAGE` backbuffer does not support the standard X11 bitmap fonts and logical pixle operations. The default back buffer is `EZ_PIXMAP`.

EZwgl also provides a function to select the back buffer automatically.

`void EZ_AutoSelectBackBuffer(void);`

This function selects a back buffer based on the environment variable `EZ_BACK_BUFFER`, which can be either `pixmap` or `ximage`. If this environment variable is not set, then it makes a selection based on the following creterion: if the server supports shared memory extension and it is usable, then it selects the `EZ_XIMAGE` back buffer; otherwise, it selects the `EZ_PIXMAP` back buffer.

To signal which buffer to draw, use

`void EZ_DrawBuffer(int buf);`

Use the next function to copy the contents of a back buffer to the front.

`void EZ_SwapBuffers(void);`

13.4.3 Choosing a Shading Model

By default, the graphics library uses a flat shading model for coloring geometric objects, i.e., all vertices are assigned the same color. A primitive can also be smooth (Gouraud) shaded. In this model, all vertices are treated individually. The library uses linear interpolation to color the primitive. Shading model is controled by

`void EZ_ShadeModel(ind mode);`

This function selects one of the two shading modes: `EZ_SMOOTH` or `EZ_FLAT`.

13.4.4 Clear Color

For each 3DCanvas, there is a clear color. It is used by `EZ_Clear` to reset the draw buffer. The next two commands set the clear color.

```
void EZ_ClearColorf(float r, float g, float b);
void EZ_ClearIndex(ind idx)
```

The next command clears the specified buffers. `void EZ_Clear(ind mode)`
Valid modes are `EZ_COLOR_BUFFER_BIT` or `EZ_DEPTH_BUFFER_BIT` or the two or'ed together.

13.4.5 Example G2, The Sine Curve

This short example demonstrates the use of double buffers. We create a 3DCanvas and draw a sine curve into it continuously. At each draw we shift the phase a little bit, creating the illusion of a moving wave.

We use a check button to turn the double buffer on/off. In double buffer mode, the animation looks smooth, while in single buffer mode, the curve flickers too much.

We also provide a slider to adjust the speed of the animation.

```
/************** Example G2 *****************************************/
#include "EZ.h"

static void eventHandler(EZ_Widget *, void *, int, XEvent *);
static void draw(EZ_Widget *);
static void sliderCallBack(EZ_Widget *, void *);
```

```
static void cbtnCallBack(EZ_Widget *, void *);
static float speed = 0.1;
static int  bufferMode = 1;

main(int argc, char **argv)
{
  EZ_Widget *frame, *canvas, *slider, *cbtn;
  /*
   * Initialize EZWGL, don't forget to
   * initialize the graphics library also.
   */
  EZ_Initialize(argc, argv, 1);      /* notice the 1        */
  /*
   * canvases do not resize propertly as toplevel
   * widgets (bug!) It is recommended to put it
   * under a frame at least for now.
   */
  frame = EZ_CreateWidget(EZ_WIDGET_FRAME, NULL,
                        EZ_FILL_MODE, EZ_FILL_BOTH,
                        0);
  canvas = EZ_CreateWidget(EZ_WIDGET_3D_CANVAS, frame,
                         EZ_ORIENTATION,   EZ_VERTICAL_BOTTOM,
                         EZ_WIDTH_HINT,    320,
                         EZ_HEIGHT_HINT,   320,
                         EZ_EVENT_HANDLER, eventHandler, NULL,
                         0);
  slider = EZ_CreateWidget(EZ_WIDGET_VERTICAL_SLIDER, frame,
                         EZ_LABEL_STRING,      "speed",
                         EZ_SLIDER_RANGE,      0.0, 10.0,
                         EZ_SLIDER_INIT_VALUE, 1.0,
                         EZ_CALLBACK,          sliderCallBack, &speed,
                         0);

  /* create a button to toggle double buffer on/off */
  cbtn = EZ_CreateWidget(EZ_WIDGET_CHECK_BUTTON, canvas,
                       EZ_LABEL_STRING,        "Turn DoubleBuffer On/Off",
                       EZ_CHECK_BUTTON_ON_VALUE,  0,
                       EZ_CHECK_BUTTON_OFF_VALUE, 1,
                       EZ_CHECK_BUTTON_ON_OFF,    EZ_ON,
                       EZ_BORDER_WIDTH,           2,
                       EZ_BORDER_TYPE,            EZ_BORDER_RAISED,
                       EZ_CALLBACK,               cbtnCallBack, NULL,
                       0);
  /*
   * now display the canvas. One must first display
   * a 3DCanvas before calling any GL functions !!!
   */
  EZ_DisplayWidget(canvas);

  /* now setup global GL attributes */
  EZ_RGBMode();                 /* we are using RGB mode  */
  EZ_AutoSelectBackBuffer();    /* select a back buffer   */
  EZ_DrawBuffer(EZ_BACK);/* always draw into the back buf */

  /*
   * We'll draw a 2D polygon, so we don't have to setup
   * a complex projection matrix. The default matrix mode
   * is EZ_MODELVIEW. We just set the projection matrix
   * on this default matrix stack.
   */
  EZ_LoadIdentity();            /* clear the top mtx 2 id */
  EZ_Ortho2(0.0,10.0,-1.3,1.3); /* project the unit cube  */

  /*
   * We don't draw anything here, let the
   * event handler handle the drawing
   */
  while(1)
    {
      EZ_ServiceEvents();
      EZ_DrawBuffer(bufferMode == 0? EZ_FRONT: EZ_BACK);
      draw(canvas);
      if(bufferMode != 0) EZ_SwapBuffers();
    }
}

static void draw(EZ_Widget *canvas)
{
  static float shift = 0.0;
  float  t, x, y;
  int    i;

  EZ_Clear(EZ_COLOR_BUFFER_BIT); /* clear the frame buffer */
  EZ_Color3f(1.0,1.0,0.0);

  EZ_Begin(EZ_LINE_STRIP);       /* start a polyline       */

  t = shift;
  shift += speed;
  x = 0.0;
```

```
  for(i = 0; i < 1000; i++)
    {
      y = sin(t+x);
      x += 0.01;

      EZ_Vertex2f(x, y);
    }
  EZ_End();
}
/*
 * the event handler. It should handle at least the
 * EZ_REDRAW event and the EZ_RESIZE.
 */
static void eventHandler(EZ_Widget *canvas, void *data,
                         int eventType, XEvent *xevent)
{
  switch(eventType)
    {
    case EZ_REDRAW:
    case EZ_RESIZE:
      draw(canvas);
      if(bufferMode != 0) EZ_SwapBuffers();
      break;
    case EZ_KEY_PRESS:
      if(EZ_PressedKey == EZ_ESCAPE_KEY) exit(0);
      break;
    default:
      break;
    }
}

static void sliderCallBack(EZ_Widget *slider, void *data)
{
  float *f = (float *)data;
  *f = 0.05 * EZ_GetSliderValue(slider);
}

static void cbtnCallBack(EZ_Widget *cbtn, void *data)
{
  EZ_GetCheckButtonState(cbtn, &bufferMode);
}
/*************** Example G2 ******************************************/
```

13.5 EZ_Enable and EZ_Disable

Most display state Attributes are controlled by this pair of routines

`void EZ_Enable(int target);`

`void EZ_Disable(int target);`

The next table lists all the targets (state attributes) controled by these two functions.

Symbolic name	default state
`EZ_LIGHTING`	disabled
`EZ_LIGHT0 -- EZ_LIGHT7`	all disabled
`EZ_NORMALIZE`	disabled
`EZ_COLOR_MATERIAL`	disabled
`EZ_CULL_FACE`	disabled
`EZ_FOG`	disabled
`EZ_DEPTH_TEST`	disabled
`EZ_DITHER`	normally disabled

13.6 Default State Attributes

The state of the graphics library is retained in the graphics context data structure. The default state attributes are listed in the following table.

Attribute	default value
Display mode	Color map
Draw buffer	front
Back buffer	undefined
Zbuffer	disabled
Dithering	disabled
Font	large vector font
Lighting	disabled
Shade Model	flat
Foging	disabled
Cull face	disabled
Clear color	(0,0,0)
Clear index	0
Matrix mode	`EZ_MODELVIEW`

13.7 Manipulating Matrices

There are two matrix stacks that transform and project geometric primitives to the screen. One matrix stack is called the `EZ_MODELVIEW`. The top matrix on this stack is used to transform the current primitive to the eye coordinate space. The other stack is called `EZ_PROJECTION`. The top matrix on this stack is used to project the transformed geometric primitives to the normalized screen space.

To set the current matrix mode, use

`void EZ_MatrixMode(int mode);`

Modes are: `EZ_MODELVIEW` or `EZ_PROJECTION`.

The $\mathcal{EZ}$ graphics library uses **row vectors**. If the top matrix on the `EZ_MODELVIEW` stack is M and the top matrix on the `EZ_PROJECTION` stack is P, then for any given vector $\overline{v}$, the transformed coordinate in the eye coordinate space is $\overline{V_e} = \overline{v}M$; and the projected coordinate in the homogeneous screen space (unclipped) is $\overline{v_p} = \overline{v}MP$. This coordinate is then clipped and scaled to the current viewport.

There are a few more commands for manipulating matrices.

`void EZ_LoadIdentity(void)`

This function resets the top matrix of the currment matrix stack to Identity.

`void EZ_LoadMatrix(EZ_Matrix M)`

This function replace the top matrix of the current matrix stack by matrix `M`.

`void EZ_MultMatrix(EZ_Matrix M)`

This function left multiply the top matrix of the current matrix stack by matrix M. If the current top matrix is T, this function replaces T with MT.

`void EZ_GetMatrix(EZ_Matrix M)`

This function copies the top matrix of the current matrix stack to the given matrix M.

`void EZ_PushMatrix(void)`

This function pushes down the current matrix stack, duplicating the top matrix.

`void EZ_PopMatrix(void)`

This function pops the current matrix stack.

The following routines are normally used to modify the projection matrices.

```
void EZ_Ortho(float l, float r, float b, float t,
            float n, float f);
```

This function defines a box-shaped encloure in the eye coordinate system. `l, r,b,t,n,f` specifies the boundries of the box, they are the clipping planes. This function multiply the current matrix by

$$\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & -\frac{f+n}{f-n} & 1 \end{pmatrix}$$

`void EZ_Ortho2(float left, float right, float bottom, float top)`

This routine sets up a 2D clipping rectangle. It is equivalent to `EZ_Ortho` with parameter `n = -1.0, f= 1.0`.

`void EZ_Perspective(float fov, float, asp, float n, float f)`

This routine defines a perspective viewing pyramid inside the eye coordinate system. The vertical direction is the y direction in the eye coordinate system. This function multiply the current matrix by

$$\begin{pmatrix} \frac{\cot\left(\frac{\text{fov}}{2}\right)}{\text{asp}} & 0 & 0 & 0 \\ 0 & \cot\left(\frac{\text{fov}}{2}\right) & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{pmatrix}$$

```
void EZ_Frustum(float l, float r, float b,
                float t, float n, float f);
```

This routine defines a perspective viewing pyramid inside the eye coordinate system. The eye is at the origin. The lower-left and upper-right corners of the near clipping window are $(l, b, -n)$ and $(r, t, -n)$. The parameter f specifies the distance from the eye to the far clipping plane. This function multiply the current matrix by

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ \frac{r+l}{r-l} & \frac{t+b}{t-b} & -\frac{f+n}{f-n} & -1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{pmatrix}$$

The following command is normally used to define the eye coordinate system.

```
void EZ_LookAt(float vx, float vy, float vz,
               float px, float py, float pz,
               float ux, float uy, float uz);
```

This routine defines a view point and a reference point on the line of sight, together with a view up vector which defines the direction of the y-axis of the eye coordinate system. This function multiply the current matrix by

Trans(-vx,-vy,-vz)×M where M is a matrix described below.

Let $\overline{\mathbf{v}} = (v_x, v_y, v_z), \overline{\mathbf{p}} = (p_x, p_y, p_z), \overline{\mathbf{u}} = (u_x, u_y, u_z)$ and define

$$\begin{aligned} \overline{\mathbf{V_3}} &= \overline{\mathbf{v}} - \overline{\mathbf{p}} \\ \overline{\mathbf{V_2}} &= \overline{\mathbf{u}} - \frac{\overline{\mathbf{u}} \cdot \overline{\mathbf{V_3}}}{||\overline{\mathbf{V_3}}||^2}\overline{\mathbf{V_3}} \\ \overline{\mathbf{V_1}} &= \overline{\mathbf{V_3}} \times \overline{\mathbf{V_1}} \end{aligned}$$

Then

$$M = \begin{pmatrix} \frac{\overline{\mathbf{V_1}}}{||\overline{\mathbf{V_1}}||} & \frac{\overline{\mathbf{V_2}}}{||\overline{\mathbf{V_2}}||} & \frac{\overline{\mathbf{V_3}}}{||\overline{\mathbf{V_3}}||} & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Equivalently, let $n = \sqrt{(v_x - p_x)^2 + (v_y - p_y)^2 + (v_z - p_z)^2}$, $d = ((v_x - p_x)u_x + (v_y - p_y)u_y + (v_z - p_z)u_z)/n$, $z_0 = (v_x - p_x)/n, z_1 = (v_y - p_y)/n, z_2 = (v_z - p_z)/n$, $t_0 = u_x - d * z_0, t_1 = u_y - d * z_1, t_2 = u_z - d * z_2$, $s = \sqrt{(t_0^2 + t_1^2 + t_2^2)}$ and $y_0 = t_0/s, y_1 = t_1/s, y_2 = t_2/s$. Then

$$M = \begin{pmatrix} y_1 z_2 - y_2 z_1 & y_0 & z_0 & 0 \\ y_2 z_0 - y_0 z_2 & y_1 & z_1 & 0 \\ y_0 z_1 - y_1 z_0 & y_2 & z_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The next few routines are general purpose commands for manipulating matrices.

```
void EZ_Translate(float x, float y, float z);
```

This command multiply the current matrix by

$$\text{Trans}(x, y, z) \quad = \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & z & 1 \end{pmatrix}$$

`void EZ_Scale(float x, float y, float z);`

This command multiply the current matrix by

$$\mathrm{Scale}(x,y,z) \quad = \quad \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

`void EZ_Rotate(float angle, float x, float y, float z);`

This command multiply the current matrix by

$$\mathrm{Rot}(\mathrm{angle},x,y,z) \quad = \quad \begin{pmatrix} & & & 0 \\ & A & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where A is the matrix

$$A = \exp \begin{pmatrix} 0 & -z' & y' \\ z' & 0 & -x' \\ -y' & x' & 0 \end{pmatrix} \mathrm{angle}$$

where $x' = x/\sqrt{x^2+y^2+z^2}$, $y' = y/\sqrt{x^2+y^2+z^2}$ and $z' = z/\sqrt{x^2+y^2+z^2}$.

```
void EZ_GenerateRotationMatrix(EZ_Matrix M,
        float angle, float x, float y, float z);
```

This command generate the rotation matrix **Rot(angle, x,y,z)** and copy the result to the given matrix M.

13.7.1 Example G3, Using Matrices

In this example, we draw a face using spheres. We use matrices to scale and translate the (same) sphere to different locations.

```
/********************* Example G3 **********************************************/
#include "EZ.h"

static void event_handle(EZ_Widget *, void *,int,XEvent *); /* event handle   */
static void draw(EZ_Widget *canvas);                        /* repaint func   */
static float light_position[] = {0.5, 0.5, 0.0, 1.0};    /* light position */
static float white[] = {1.0,1.0,1.0,1.0};
static float shininess[] = {8.0};

main(int argc, char **argv)
{
  EZ_Widget *frame, *canvas;

  EZ_Initialize(argc, argv, 1);

  frame = EZ_CreateWidget(EZ_WIDGET_FRAME, NULL,
                         EZ_FILL_MODE, EZ_FILL_BOTH,
                         0);
  canvas = EZ_CreateWidget(EZ_WIDGET_3D_CANVAS, frame,
                          EZ_WIDTH_HINT,    320,
                          EZ_HEIGHT_HINT,   320,
                          EZ_EVENT_HANDLER, event_handle, NULL,
                          0);

  EZ_DisplayWidget(canvas);

  /* setup GL modes */
  EZ_AutoSelectBackBuffer();                          /* select a back buffer */
  EZ_RGBMode();                                       /* select color mode    */
  EZ_ShadeModel(EZ_SMOOTH);                           /* select shading model */
  EZ_DrawBuffer(EZ_BACK);                             /* draw into the backbuf*/
  EZ_Enable(EZ_CULL_FACE);                            /* enable backface cull */
  EZ_Enable(EZ_DEPTH_TEST);                           /* enable zbuffer       */
  EZ_Enable(EZ_LIGHTING);                             /* enable lighting      */
  EZ_Enable(EZ_LIGHT0);                               /* enable one light src */

  EZ_Lightfv(EZ_LIGHT0, EZ_POSITION, light_position);
  EZ_Materialfv(EZ_FRONT,EZ_SPECULAR,white);   /* set material for front face */
  EZ_Materialfv(EZ_FRONT,EZ_SHININESS,shininess);

  EZ_MatrixMode(EZ_PROJECTION);           /* setup a fixed projection matrix  */
  EZ_LoadIdentity();
```

```
  EZ_Perspective(45.0, 1.0, 2.0, 30.0);   /* a perspective view, ...          */
  EZ_MatrixMode(EZ_MODELVIEW);            /* set matrix mode to model view    */
  EZ_LoadIdentity();
  EZ_LookAt(0.3,0.1,15.0, 0.0,0.0,0.0,    /* and set up a look at view        */
           0.0,1.0, 0.0);                 /* view up vector                   */

  EZ_EventMainLoop();                     /* turn control to EZWGL            */
}

static float rotX = 0.0, rotY = 0.0, rotZ = 0.0;      /* rotation angles      */
static float yellow[] = {0.8, 0.8, 0.0, 0.0};         /* material color       */
static float red[] = {0.8, 0.4, 0.3, 0.0};            /* material color       */

static void draw(EZ_Widget *canvas)
{
  EZ_Clear(EZ_COLOR_BUFFER_BIT | EZ_DEPTH_BUFFER_BIT);

  EZ_PushMatrix();
  EZ_Rotate(rotX, 1.0, 0.0, 0.0 );
  EZ_Rotate(rotY, 0.0, 1.0, 0.0 );
  EZ_Rotate(rotZ, 0.0, 0.0, 1.0 );

  /* draw a red sphere */
  EZ_Materialfv(EZ_FRONT,EZ_DIFFUSE, red);
  EZ_Sphere(EZ_SPHERE_TRIANGLE, 4, 0.0,0.0,0.0,3.0);

  /* translate to (-1.0, 0.8, 0.8), scale down unformly by 0.2
   * and draw a yellow sphere. */
  EZ_PushMatrix();
  EZ_Materialfv(EZ_FRONT,EZ_DIFFUSE, yellow);
  EZ_Translate(-1.0,0.8,2.2);
  EZ_Scale(0.2,0.2,0.2);
  EZ_Sphere(EZ_SPHERE_TRIANGLE, 4, 0.0,0.0,0.0,3.0);
  EZ_PopMatrix();

  /* translate to (1.0, 0.8, 0.8), scale down unformly by 0.2
   * and draw a yellow sphere */
  EZ_PushMatrix();
  EZ_Materialfv(EZ_FRONT,EZ_DIFFUSE, yellow);
  EZ_Translate(1.0,0.8,2.2);
  EZ_Scale(0.2,0.2,0.2);
  EZ_Sphere(EZ_SPHERE_TRIANGLE, 4, 0.0,0.0,0.0,3.0);
  EZ_PopMatrix();

  /* and so on */
  EZ_PushMatrix();
  EZ_Materialfv(EZ_FRONT,EZ_DIFFUSE, yellow);
  EZ_Translate(0.0,-0.1,2.65);
  EZ_Scale(0.15,0.35,0.15);
  EZ_Sphere(EZ_SPHERE_TRIANGLE, 4, 0.0,0.0,0.0,3.0);
  EZ_PopMatrix();

  EZ_PushMatrix();
  EZ_Materialfv(EZ_FRONT,EZ_DIFFUSE, yellow);
  EZ_Translate(0.0,-1.6,2.6);
  EZ_Scale(0.4,0.1,0.1);
  EZ_Sphere(EZ_SPHERE_TRIANGLE, 4, 0.0,0.0,0.0,3.0);
  EZ_PopMatrix();

  EZ_PopMatrix();
  EZ_SwapBuffers();                                 /* copy backbuf to front  */
}

static void event_handle(EZ_Widget *canvas, void *data, int eventType, XEvent *xev)
{
  switch(eventType) {
  case EZ_REDRAW:
  case EZ_RESIZE:
    draw(canvas);
    break;
  case EZ_KEY_PRESS:
    switch(EZ_PressedKey) {
    case EZ_DOWN_KEY:   rotX += 5.0; draw(canvas); break;
    case EZ_UP_KEY:     rotX -= 5.0; draw(canvas); break;
    case EZ_RIGHT_KEY:  rotY += 5.0; draw(canvas); break;
    case EZ_LEFT_KEY:   rotY -= 5.0; draw(canvas); break;
    case EZ_ESCAPE_KEY:   EZ_Shutdown(); exit(0);  break;
    default:                                       break;
    }
    EZ_GLResetEvents(NULL); /* clear the event queue, too many keypresses */
    break;
  default:
    break;
  }
}
/******************** Example G3 *********************************************/
```

13.8 Specifying Geometric Objects

Geometric objects are specified by enclosing a series of vertices, normals and colors between `EZ_Begin` and `EZ_End` pair.

13.8.1 EZ_Begin and EZ_End

This pair of routines provides the only way to specify geometrical primitives.

```
void EZ_Begin(int PType);
void EZ_End(void);
```

Possible primitive types are:

```
EZ_TRIANGLE_STRIP
EZ_TRIANGLE_FAN
EZ_TRIANGLES
EZ_QUAD_STRIP
EZ_QUADS
EZ_POLYGON /* buggy */
EZ_LINE_STRIP
EZ_LINES
EZ_LINE_LOOP
EZ_POINTS
EZ_TMESH (rectangular triangular mesh, row majored)
EZ_QMESH (rectangular quadrilateral mesh, row majored)
```

For the last two primitive types, one must call `void EZ_SetTQMeshSize` right after `EZ_Begin`. This function informs the graphical library about the size of the triangular or rectangular mesh being processed.

```
void EZ_SetTQMeshSize(int row, int col);
```

The only GL calls that are allowed inside an `EZ_Begin` and `EZ_End` pair are

- `EZ_Vertex{23}f{v}`
- `EZ_Normal3f{v}`
- `EZ_Color3f{v}`
- `EZ_Index`
- `EZ_CallList{s}`
- `EZ_Materialf{v}`

13.8.2 A Brief Description of Geometrical Primitives

- `EZ_TRIANGLE_STRIP` A triangle strip is series of triangles connected along shared edges. It is defined by a series of vertices. The first three vertices define the first triangle. Each subsequent vertex defines a new triangle using that vertex along with two vertices from the previous triangle. For example, if v_1, v_2, v_3, v_4, v_5 are the vertices specified between the `EZ_Begin(EZ_TRIANGLE_STRIP)` and `EZ_End()`, then it defines three triangles, $< v_1, v_2, v_3 >$, $< v_2, v_3, v_4 >$ and $< v_3, v_4, v_5 >$.

- `EZ_TRIANGLE_FAN` A triangle fan is a series of triangles share all share the same first vertex. For example, if v_1, v_2, v_3, v_4, v_5 are the vertices specified between the `EZ_Begin(EZ_TRIANGLE_FAN)` and `EZ_End()`, then it defines three triangles, $< v_1, v_2, v_3 >$, $< v_1, v_3, v_4 >$ and $< v_1, v_4, v_5 >$.

- `EZ_TRIANGLES` This primitive specifies a series of individual triangles. The triangles does not share vertices. For example, if $v_1, v_2, v_3, v_4, v_5, v_6$ are the vertices between the `EZ_Begin(EZ_TRIANGLES)` and `EZ_End()`, then it defines two triangles, $< v_1, v_2, v_3 >$ and $< v_4, v_5, v_6 >$.

- `EZ_QUAD_STRIP` This primitive specifies a quadrilateral strip, a series of edge-sharing quadrilaterals. The first 4 vertices define the first quad. Each subsequent two vertices define a new quad using the last two vertices and the two new vertices, with the order of the two new vertices swapped. For example, the sequence $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$ define three quads: $[v_1, v_2, v_3, v_4]$, $[v_3, v_4, v_6, v_5]$ and $[v_5, v_6, v_8, v_7]$.

- `EZ_QUADS` This primitive specifies a series of individual quads. The quads do not share vertices. For example, the sequence $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$ define two quads: $[v_1, v_2, v_3, v_4]$ and $[v_5, v_6, v_7, v_8]$.

- `EZ_LINE_STRIP` This primitive defines a polyline.

- `EZ_LINES` This primitive defines a series of line segments. For example, the sequence $v_1, v_2, v_3, v_4, v_5, v_6$ define three line segments, $\overline{v_1v_2}$, $\overline{v_3v_4}$ and $\overline{v_5v_6}$.
- `EZ_LINE_LOOP` This primitive defines a closed polyline. It is the same as `EZ_LINE_STRIP` with the first vertex appended to the list of specified vertices.
- `EZ_POINTS` This primitive specifies a sequence of points.

13.8.3 EZ_Vertex{23}f{v} and EZ_Normal3f{v}

`EZ_Vertex{23}f{v}` specifies the coordinates of a vertex and `EZ_Normal3f{v}` specifies the normal vector at a vertex. Arguments are either 2 floats or 3 floats or a pointer to 2 or 3 floats. The normal vector is used in lighting calculations. For example, the following sequence specifies a rectangle.

```
EZ_Begin(EZ_QUADS);          /* begin individual quads   */
EZ_Normal3f(1.0, 0.0, 0.0);  /* set the normal           */
EZ_Vertex2f(0.0, 1.0);       /* the first vertex         */
EZ_Vertex2f(1.0, 1.0);
EZ_Vertex2f(1.0, 0.0);
EZ_Vertex2f(0.0, 1.0);
EZ_End();                   /* end the current object  */
```

13.8.4 Pixmaps

Support for pixmaps has been added in release 1.38. To display a pixmap/bitmap on a 3DCanvas, one first call `EZ_CreageGLImage` to create an image from *EZwgl* pixmap (created by `EZ_CreateLabelPixmapFrom***`), then use `EZ_PutGLImage***` to display the image in the current draw buffer. Relevent routines are the following.

```
EZ_GLImage *EZ_CreateGLImage(EZ_Pixmap *pix);
void EZ_GetGLImageWindow(int x, int y, int w, int h, EZ_GLImage **imgRet);
void EZ_GetGLImageXWindow(int x, int y, int w, int h, EZ_GLImage **imgRet);

void EZ_GetGLImageInfo(EZ_GLImage *glimage, Pixmap *pix_ret,
                       Pixmap *shape_ret, int w_ret, int h_ret);

void EZ_FreeGLImage(EZ_GLImage *glimage);

EZ_Pixmap *EZ_GetLabelPixmapFromGLImage(EZ_GLImage *image);

void EZ_PutGLImage(EZ_GLImage *glimage,
                   int srcX, int srcY, int srcW, int srcH,
                    float destX, float destY, float destZ);
void EZ_PutGLImageScreen(EZ_GLImage *glimage,
                    int srcX, int srcY, int srcW, int srcH,
                    float destX, float destY, float destZ);
void EZ_PutGLImageWindow(EZ_GLImage *glimage,
                    int srcX, int srcY, int srcW, int srcH,
                    int destX, int destY);
void EZ_PutGLImageXWindow(EZ_GLImage *glimage,
                    int srcX, int srcY, int srcW, int srcH,
                    int destX, int destY);
```

13.9 Lighting

Lighting is controled by three items: light sources, light model and material properties. The color of a vertex is calculated based on the following factors: (a) the position of the vertex, the current normal vector and the current material properties. (b) the position and properties of the light sources and (c) the light model properties.

For a complete description of lighting calculations, please see

```
AUTHOR: Neider, Jackie
TITLE: OpenGL programming guide : the official guide to learning
      OpenGL, release 1 /OpenGL Architecture Review Board, Jackie
                      Neider, Tom Davis, Mason Woo.
PUBLISHED:      Reading, Mass. : Addison-Wesley, c1993.
DESCRIPTION:    xxxiii, 516 p., (16) p. of plates : ill. (some col.) ; 24 cm.
NOTES:          Includes index.
```

```
SUBJECTS:       OpenGL
                Computer graphics
OTHER AUTHORS:  Davis, Tom
                Woo, Mason
                OpenGL Architecture Review Board
ISBN:           0201632748
OCLC NUMBER:    27679041
```

To use lighting, follow the following steps.

- Call `EZ_Enable(EZ_LIGHTING)` to enable lighting calculations.
- Call `EZ_Enable(EZ_LIGHT#)` to turn on light sources, where `#` is an integer between 0 and 7. One can turn on multiple light sources.
- Call `EZ_Lightfv`, `EZ_LightModelfv` and `EZ_Materialfv` to set light source properties, light model properties and material properties.
- When define a geometric object, make sure you set the normal vectors by using `EZ_Normal3f{v}`. If needed, one may enable color material and use `EZ_Color3f{v}` to set vertex colors.

13.9.1 Example G4, Lighting

Here is a short example that uses lighting calculations. In this example, we turn on one light source, set its positon but use its default color. We set the local viewer property of the light model. Then we render a yellow dodecahedron and a red sphere.

```
/******************** Example G4 ********************************************/
#include ``EZ.h''

static void event_handle(EZ_Widget *, void *,int,XEvent *); /* event handle   */
static void draw(EZ_Widget *canvas);                        /* repaint func   */
static float light_position[] = {0.5, 0.5, 0.0, 1.0};       /* light position */
static float white[] = {1.0,1.0,1.0,1.0};
static float shininess[] = {8.0};

main(int argc, char **argv)
{
  EZ_Widget *frame, *canvas;

  /* create a canvas */
  EZ_Initialize(argc, argv, 1);

  frame = EZ_CreateWidget(EZ_WIDGET_FRAME, NULL,
                       EZ_FILL_MODE, EZ_FILL_BOTH,
                       0);
  canvas = EZ_CreateWidget(EZ_WIDGET_3D_CANVAS, frame,
                         EZ_ORIENTATION,   EZ_VERTICAL_BOTTOM,
                         EZ_WIDTH_HINT,    320,
                         EZ_HEIGHT_HINT,   320,
                         EZ_EVENT_HANDLER, event_handle, NULL,
                         0);

  EZ_DisplayWidget(canvas);

  /* setup GL modes */
  EZ_AutoSelectBackBuffer();                          /* select a back  buffer */
  EZ_RGBMode();                                       /* select color  mode    */
  EZ_ShadeModel(EZ_SMOOTH);                           /* select  shading model */
  EZ_DrawBuffer(EZ_BACK);                             /* draw into the  backbuf*/
  EZ_Enable(EZ_CULL_FACE);                            /* enable  backface cull */
  EZ_Enable(EZ_DEPTH_TEST);                           /* enable  zbuffer       */

  /* lighting related attributes. One should be aware
   * that where to invoke EZ_Lightfv or EZ_Materialfv
   * matters a lot. The corresponding properties are
   * computed based on the current model view matrix.
   */
  EZ_Enable(EZ_LIGHTING);                             /* enable lighting      */
  EZ_Enable(EZ_LIGHT0);                               /* enable one light src */
  /* set light src pos, this light src has a fixed position, i.e.
   * the position will not be affected by the model view matrux */
  EZ_Lightfv(EZ_LIGHT0, EZ_POSITION, light_position);
  EZ_Materialfv(EZ_FRONT,EZ_SPECULAR,white);   /* set material for front face */
  EZ_Materialfv(EZ_FRONT,EZ_SHININESS,shininess);
  EZ_LightModelfv(EZ_LIGHT_MODEL_LOCAL_VIEWER, white);

  EZ_MatrixMode(EZ_PROJECTION);           /* setup a fixed projection matrix */
  EZ_Perspective(60.0, 1.0, 2.0, 30.0);   /* a perspective view, ...         */
  EZ_MatrixMode(EZ_MODELVIEW);            /* set matrix mode to model view   */
  EZ_LookAt(0.0,0.0,15.0, 0.0,0.0,0.0,    /* and set up a look at view       */
          0.0,1.0, 0.0);                  /* view up vector        */
```

```
  EZ_EventMainLoop();                     /* turn control to EZWGL           */
}

static float rotX = 0.0, rotY = 0.0, rotZ = 0.0;      /* rotation angles      */
static float scale = 1.0;                             /* zooming factor       */
static float yellow[] = {0.8, 0.8, 0.0, 0.0};         /* material color       */
static float red[] = {0.8, 0.1, 0.0, 0.0};            /* material color       */

static void draw(EZ_Widget *canvas)
{
  /*
   * since we only have one 3DCanvas, which is always
   * active, we actually don't need the argument.
   */
  EZ_Clear(EZ_COLOR_BUFFER_BIT |                    /* clear the frame buffer */
           EZ_DEPTH_BUFFER_BIT);                    /* and the zbuffer        */
  EZ_PushMatrix();
  EZ_Scale(scale,scale,scale);                      /* a uniform scale        */
  EZ_Rotate(rotX, 1.0, 0.0, 0.0 );                  /* rotate about X, Y & Z  */
  EZ_Rotate(rotY, 0.0, 1.0, 0.0 );
  EZ_Rotate(rotZ, 0.0, 0.0, 1.0 );
  EZ_PushMatrix();
  EZ_Translate( -3.0, 0.0, 0.0 );
  EZ_Materialfv(EZ_FRONT,EZ_DIFFUSE, yellow);      /* a yellow material */
  EZ_Dodecahedron(0.0,0.0,0.0,2.4);
  EZ_PopMatrix();
  EZ_PushMatrix();
  EZ_Translate(3.0,0.0,0.0);
  EZ_Materialfv(EZ_FRONT,EZ_DIFFUSE, red);         /* a red material  */
  EZ_Sphere(EZ_SPHERE_TRIANGLE, 4,0.0,0.0,0.0,2.0);
  EZ_PopMatrix();
  EZ_PopMatrix();
  EZ_SwapBuffers();
}

static void event_handle(EZ_Widget *canvas, void *data, int eventType, XEvent *xev)
{
  switch(eventType) {
  case EZ_REDRAW:
  case EZ_RESIZE:
    draw(canvas);
    break;
  case EZ_KEY_PRESS:
    switch(EZ_PressedKey) {
    case EZ_DOWN_KEY:   rotX += 5.0; draw(canvas); break;
    case EZ_UP_KEY:     rotX -= 5.0; draw(canvas); break;
    case EZ_RIGHT_KEY:  rotY += 5.0; draw(canvas); break;
    case EZ_LEFT_KEY:   rotY -= 5.0; draw(canvas); break;
    case 'e':           scale *= 1.1;draw(canvas); break;
    case 'r':           scale *= 0.9;draw(canvas); break;
    case EZ_ESCAPE_KEY:
      EZ_DestroyWidget(canvas);
      EZ_Shutdown();
      exit(0);
      break;
    default:
      break;
    }
    EZ_GLResetEvents(NULL); /* clear the event queue, too many keypresses */
    break;
  default:
    break;
  }
}
/********************* Example G4 *********************************************/
```

13.9.2 Setting Light Source Properties

A light source has the following properties.

- `EZ_AMBIENT` ambient color.
- `EZ_DIFFUSE` diffuse color.
- `EZ_SPECULAR` specular color.
- `EZ_POSITION` position.
- `EZ_SPOT_DIRECTION` spot direction for spot light source.
- `EZ_SPOT_EXPONENT` spot exponent.
- `EZ_SPOT_CUTOFF` spot cutoff angle.
- `EZ_CONSTANT_ATTENUATION` constant attenuation factor.

- `EZ_LINEAR_ATTENUATION` linear attenuation coefficient.
- `EZ_QUADRATIC_ATTENUATION` quadratic attenuation coefficient.

All these properties are set by the function

```
void EZ_Lightfv(int lightNum, int propName, float *prop)
```

The default light sources properties are list below.

ambient color	(0.0, 0.0, 0.0)
diffuse color for `LIGHT0` diffuse color for other lig srcs	(1.0,1.0,1.0) (0.0,0.0,0.0)
specular color for `LIGHT0` specular color for other lig srcs	(1.0,1.0,1.0) (0.0,0.0,0.0)
position	(0.0,0.0,1.0,0.0)
spot direction	(0.0,0.0,-1.0)
spot exponent	0.0
spot cutoff angle	180.0°
constant attenuation factor	1.0
linear attenuation coefficient	0.0
quadratic attenuation coefficient	0.0

13.9.3 Setting Light Model Properties

Each graphics context has a unique light model. A light model has the following properties.

- `EZ_LIGHT_MODEL_AMBIENT`
- `EZ_LIGHT_LOCAL_VIEWER`
- `EZ_LIGHT_TWO_SIDE`

These properties are set by

```
void EZ_LightModelfv(int propName, float *prop);
```

Here are the default light model properties.

ambient color	(0.2, 0.2, 0.2)
local viewer	0.0
tow side mode	0.0

13.9.4 Setting Material Properties

Each material has the following properties.

- `EZ_EMISSION` emissive color.
- `EZ_AMBIENT` ambient color.
- `EZ_DIFFUSE` diffuse color.
- `EZ_SPECULAR` specular color.
- `EZ_SHININESS` specular exponent.
- `EZ_COLOR_INDICES` indices for lighting on color map mode.

These properties are set by

```
void EZ_Materialfv(int face, int propName, float *prop);
```

Here are the default material properties.

emissive color	(0.0, 0.0, 0.0)
ambient color	(0.2, 0.2, 0.2)
diffuse color	(0.8, 0.8, 0.8)
specular color	(0.0, 0.0, 0.0)
shininess	0.0
color indices	0.0 1.0 1.0

13.9.5 Convinence Routines

The *EZwgl* provides a few handy routines for packing lighting properties together. They can be used to save a few `EZ_***fv` calls.

`EZ_Material *EZ_DefineMaterial(float *definitions)`
This function define a material from the given defintion and return a pointer to the allocated material structure.

`void EZ_BindMaterial(int face, EZ_Material *mat);`
This function bind the defined material `mat` to `face`.
Here is an example.

```
static float mat[] = {
  EZ_SPECULAR_F,   1.0,1.0,1.0,1.0,
  EZ_DIFFUSE_F, 0.8, 0.0, 0.8, 0.0
  EZ_NULL_F,
};

EZ_Material material = EZ_DefineMaterial(mat);
... ...
EZ_BindMaterial(EZ_FRONT_AND_BACK, material);
... ...
```

Predefined Materials The *EZwgl* includes a few predefined materials. They can be retrieved via

`EZ_Material *EZ_GetDefaultMaterial(int which);` Here is a list of the predefined materials.

Symbolic Name	property values
`EZ_SILVER`	emission (0.0,0.0,0.0,1.0) ambient (0.19125, 0.0735, 0.0225, 1.0) diffuse (0.7038, 0.27048, 0.0828, 1.0) specular (0.256777, 0.137622, 0.086014, 1.0) shininess 1.0 color indices 0.0, 0.0, 1.0
`EZ_SILVER_S`	emission (0.0,0.0,0.0,1.0) ambient (0.23125, 0.23125, 0.23125, 1.0) diffuse (0.2775, 0.2775, 0.2775, 1.0) specular (0.773911, 0.773911, 0.773911, 1.0) shininess 7.0 color indices 0.0, 0.0, 1.0
`EZ_GOLD`	emission (0.0,0.0,0.0,1.0) ambient (0.24725, 0.1995, 0.0745, 1.0) diffuse (0.75164, 0.60648, 0.22648, 1.0) specular (0.628281, 0.555802, 0.366065, 1.0) shininess 4.0 color indices 0.0, 0.0, 1.0
`EZ_GOLD_S`	emission (0.0,0.0,0.0,1.0) ambient (0.24725, 0.2245, 0.0645, 1.0) diffuse (0.34615, 0.3143, 0.0903, 1.0) specular (0.797357, 0.723991, 0.208006, 1.0) shininess 6.5 color indices 0.0, 0.0, 1.0
`EZ_BRASS`	emission (0.0,0.0,0.0,1.0) ambient (0.329412, 0.223529, 0.027451, 1.0) diffuse (0.780392, 0.568627, 0.113725, 1.0) specular (0.992157, 0.941176, 0.807843, 1.0) shininess 2.2 color indices 0.0, 0.0, 1.0
`EZ_COPPER`	emission (0.0,0.0,0.0,1.0) ambient (0.19125, 0.0735, 0.0225, 1.0) diffuse (0.7038, 0.27048, 0.0828, 1.0) specular (0.256777, 0.137622, 0.086014, 1.0) shininess 1.0 color indices 0.0, 0.0, 1.0
`EZ_COPPER_S`	emission (0.0,0.0,0.0,1.0) ambient (0.2295, 0.08825, 0.0275, 1.0) diffuse (0.5508, 0.2118, 0.066, 1.0) specular (0.580594, 0.223257, 0.0695701, 1.0) shininess 4.1 color indices 0.0, 0.0, 1.0

Symbolic Name	property values
`EZ_BRONZE`	emission (0.0,0.0,0.0,1.0) ambient (0.2125, 0.1275, 0.054, 1.0) diffuse (0.714, 0.4284, 0.18144, 1.0) specular (0.393548, 0.271906, 0.166721, 1.0) shininess 2.2 color indices 0.0, 0.0, 1.0
`EZ_RUBY`	emission (0.0,0.0,0.0,1.0) ambient (0.1745, 0.01175, 0.01175, 1.0) diffuse (0.61424, 0.04136, 0.04136, 1.0) specular (0.727811, 0.626959, 0.626959, 1.0) shininess 6.5 color indices 0.0, 0.0, 1.0
`EZ_TURQUOISE`	emission (0.0,0.0,0.0,1.0) ambient (0.1, 0.18725, 0.1745, 1.0) diffuse (0.396, 0.74151, 0.69102, 1.0) specular (0.297254, 0.30829, 0.306678, 1.0) shininess 1.0 color indices 0.0, 0.0, 1.0
`EZ_JADE`	emission (0.0,0.0,0.0,1.0) ambient (0.135, 0.2225, 0.1575, 1.0) diffuse (0.54, 0.89, 0.63, 1.0) specular (0.316228, 0.316228, 0.316228, 1.0) shininess 1.2 color indices 0.0, 0.0, 1.0
`EZ_EMERALD`	emission (0.0,0.0,0.0,1.0) ambient (0.0215, 0.1745, 0.0215, 1.0) diffuse (0.07568, 0.61424, 0.07568, 1.0) specular (0.633, 0.727811, 0.633, 1.0) shininess 6.4 color indices 0.0, 0.0, 1.0
`EZ_PERAL`	emission (0.0,0.0,0.0,1.0) ambient (0.25, 0.20725, 0.20725, 1.0) diffuse (1.0, 0.829, 0.829, 1.0) specular (0.296648, 0.296648, 0.296648, 1.0) shininess 1.0 color indices 0.0, 0.0, 1.0

`EZ_LightSrc *EZ_DefineLight(float *definitions)`
This function define a light source from the given defintion and return a pointer to the allocated light source structure.

`void EZ_BindLight(int lightNum, EZ_LightSrc *lsrc)` This function bind the packed light source definition `lsrc` to light number `lightNum`.
Here is an example.

```
static float  light2[] =
{
  EZ_POSITION_F,   0.0,5.0,10.0,1.0,
  EZ_SPECULAR_F,   1.0,1.0,1.0,1.0,
  EZ_SPOT_DIRECTION_F, 0.0,0.0,-1.0,0.0,
  EZ_SPOT_CUTOFF_F,  30.0,
  EZ_SPOT_EXPONENT_F, 10.0,
  EZ_NULL_F,
};
EZ_LightSrc *lsrc2 = EZ_DefineLight(light2);
... ...
EZ_BindLight(EZ_LIGHT4, lsrc2);
... ...
```

`EZ_LightModel *EZ_DefineLightModel(float *definitions)` This function define a packed light model from the given defintion and return a pointer to the allocated light model structure.

`void EZ_BindLightModel(EZ_LightModel *lmodel)`

This function replace the current light model by the packed light model `lmodel`.

Here is an example.

```
static float  lmodelDef[] =
{
  EZ_AMBIENT_F,   0.0,0.4,0.4,0.0,
  EZ_LIGHT_MODEL_LOCAL_VIEWER, 1.0,
  EZ_LIGHT_MODEL_TWO_SIDE, 1.0,
  EZ_NULL_F,
};
EZ_LightModel *lmodel = EZ_DefineLightModel(lmdelDef);
... ...
EZ_BindLightModel(lmodel);
... ...
```

13.10 ColorMaterial

ColorMaterial is a mechanism that allows one to attach material properties to the current color, so that the library track continuously the components of its component values. ColorMaterial is one of the global state attributes and is enabled or disabled by `EZ_Enable(EZ_COLOR_MATERIAL)` or `EZ_Disable(EZ_COLOR_MATERIAL)`. It is disabled by default.

`void EZ_ColorMaterial(int face, int propName);`

This function attaches a material property `propName` to the current color. For example, if colorMaterial is enabled, the command

`EZ_ColorMaterial(EZ_FRONT, EZ_DIFFUSE)`

set the diffuse color of the front material to the current color.

It has no effect if colorMaterial is disabled.

13.11 Logical Operations

The *EZwgl* supports logical operations when drawing into the front buffer or a `EZ_PIXMAP` back buffer. Support for logical operations in `EZ_XIMAGE` back buffer is not yet implemented.

The default logical operation is `EZ_COPY`.

`void EZ_LogicOp(int op)`

This function sets the current logical operation. Possible logical operations are listed in the following table. In this table, `s` is the incoming pixel value and `d` is the pixel value stored in the framebuffer or the pixmap.

Symbolic Name	Operation
`EZ_CLEAR`	0
`EZ_AND`	$s \wedge d$
`EZ_AND_REVERSE`	$s \wedge \neg d$
`EZ_COPY`	s
`EZ_AND_INVERTED`	$\neg s \wedge d$
`EZ_NOOP`	d
`EZ_XOR`	s `xor` d
`EZ_OR`	$s \vee d$
`EZ_NOR`	$\neg(s \vee d)$
`EZ_EQUIV`	¬(s `xor` d)
`EZ_INVERT`	$\neg d$
`EZ_OR_REVERSE`	$s \vee \neg d$
`EZ_COPY_INVERTED`	$\neg s$
`EZ_OR_INVERTED`	$\neg s \vee d$
`EZ_NAND`	$\neg(s \wedge d)$
`EZ_SET`	1

13.12 Fog

Fog uses the current blending factor to blends the current fog color with the current color and use the result to color the current vertex. Fog is enabled and disabled with `EZ_Enable(EZ_FOG)` and `EZ_Disable(EZ_FOG)`. It is disabled by default.

Blending factor is computed by one of the following equations, depending on the current `fogMode`.

`fogMode`	blending factor
`EZ_LINEAR`	$f = \frac{\text{fogEnd} - (z+1)/2}{\text{fogEnd} - \text{fogStart}}$
`EZ_EXP`	$f = \exp\left(-\text{fogDensity} \cdot ((z+1) - \text{fogStart})\right)$
`EZ_EXP2`	$f = \exp\left(-\left(\text{fogDensity} \cdot ((z+1) - \text{fogStart})\right)^2\right)$

where z is the the z-coor of the vertex in the normalized screen space, $0.0 \leq$ `fogStart` $<$ `fogEnd` ≤ 1.0, `fogDensity` ≥ 0.0 are state attributes.

If fog is enabled, the color of a vertex is computed using the formula

$$\text{rgb} = \text{blendingFactor} * \text{rgb}_{\text{current}} + (1 - \text{blendingFactor}) * \text{rgb}_{\text{fog}}$$

`void EZ_Fogfv(int attrName, float *values);`

This function sets the fog state attributes. Attribute is one of

```
EZ_FOG_MODE, EZ_FOG_START, EZ_FOG_END,
EZ_FOG_DENSITY, EZ_FOG_COLOR,EZ_FOG_INDEX+
```

If `attrName` is `EZ_FOG_MODE`, the attribute value must be one of `EZ_LINEAR`, `EZ_EXP` and `EZ_EXP2`. is `EZ_FOG_START` or `EZ_FOG_END`, the attribute value must be a floating point number in $(0,1)$. If `attrName` is `EZ_FOG_INTENSITY`, the value must be a positive floating point number. If `attrName` is `EZ_FOG_COLOR`, the values must be RGB tripple, i.e., three floating point numbers in $(0,1)$ that specify the RGB intensity of the fog color.

13.13 Text

The *EZwgl* provides support for bitmap fonts and vector fonts. For `EZ_XIMAGE` back buffers, only the vector font is supported. The library also provides two methods to specify the location to render text strings: in modeling space and in screen space. To set the current font use

`void EZ_SetVectorFont(int vfont)`

This function sets the current vector font. Only two vector fonts are supported, a small vector font `EZ_SMALL_VECTOR_FONT` or a large vector font `EZ_LARGE_VECTOR_FONT`.

The next function sets the current bitmap font.

`void EZ_SetBitmapFont(char *fontName);`

To specify where to render text on the screen. Use the following set of functions.

```
void EZ_CMove3f(float x, float y, float z);
void EZ_CMove2f(float x, float y);
void EZ_CMove3fv(float xyz[3]);
void EZ_CMoveScreen3f(float x, float y, float z);
void EZ_CMoveScreen2f(float x, float y);
```

These functions set the current character position.

To draw a string, use

`void EZ_DrawString(int whichFont, char *str);`

This function draws `str` using the current vector or bitmap font at the current character position.

If the current draw buffer is an `EZ_XIMAGE` back buffer, bitmap fonts defaults to the current vector font. There is no way to draw a bitmap font to an `EZ_XIMAGE` back buffer.

13.14 Polyhedra, Sphere and Tube

The *EZwgl* includes a few functions for generating regular polyhedra, sphere and tube.

13.14.1 Tetrahedron

`void EZ_Tetrahedron(float x, float y, float z, float size);`

This function renders a tetrahedron based on the given size and center. It uses the current polygon mode to decide whether to render all face polygons or just to render the edges.

13.14.2 Cube

`void EZ_Cube(float x, float y, float z, float size);`

This function renders a cube based on the given size and center. It uses the current polygon mode to decide whether to render all face polygons or just to render the edges.

13.14.3 Octahedron

`void EZ_Octahedron(float x, float y, float z, float size);`

This function renders an octahedron based on the given size and center. It uses the current polygon mode to decide whether to render all face polygons or just to render the edges.

13.14.4 Dodecahedron

`void EZ_Dodecahedron(float x, float y, float z, float size);`

This function renders a dodecahedron based on the given size and center. It uses the current polygon mode to decide whether to render all face polygons or just to render the edges.

13.14.5 Icosahedron

`void EZ_Icosahedron(float x, float y, float z, float size);`

This function renders an icosahedron based on the given size and center. It uses the current polygon mode to decide whether to render all face polygons or just to render the edges.

13.14.6 Sphere

`void EZ_Sphere(int type, int level, float x, float y, float z, float r);`

This function renders a sphere based on the given information. It uses the current polygon mode to decide whether to render all face polygons or just to render the edges. Types are:

```
EZ_SPHERE_TRIANGLE
EZ_SPHERE_QUAD
EZ_SPHERE_PARAMETRIC
```

13.14.7 Tube

`void EZ_Tube(float *data, int nsegs, int nsides)`

This function renders a tube based on the given data. `data` specifies a **static** array of floats. It should contain `4 * nsegs` numbers. If we think of `data` as an array of 4 vectors, then the first three floats of each vector specifies a point and the last one specifies a radius. `nsegs` specifies the number of segments in `data`. `nsides` specifies the number of sides for each radial section.

For example, the following code draws a cone.

```
static float cone[] = {
-1.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.5,
 1.0, 0.0, 0.0, 1.0,
};
EZ_Tube(cone, 3, 16);
```

13.15 Polygon Modes

The graphics context has a few state variables that control how polygons are rendered. These variables are set by the following functions.

`void EZ_FrontFace(int orien);`

This function sets the front face for polygons (either `EZ_CW` or `EZ_CCW`). Each polygon has two possible faces, front and back. By default, the face with the counter-clock orientation is the front face. This function can be used to override the default

`void EZ_CullFace(int face);`

This function has no effect if cull face (back face removing) is disabled (default). The set mode is used to determine whether or not a particular polygon is rendered. The default mode is `EZ_BACK`, i.e., back facing polygons are not rendered if cull face is enabled. `EZ_Enable(EZ_CULL_FACE)` enables back face culling.

`void EZ_PolygonMode(int face, int mode);`

This function sets the rasterizing method for the `face` facing polygons. For example, the effect of `EZ_PolygoMode(EZ_FRONT,EZ_` is: for all front facing polygons only the edges are rasterized.

`face` specifies a face, either `EZ_FRONT` or `EZ_BACK` or `EZ_FRONT_AND_BACK`.

`mode` specifies a mode, either `EZ_FILL` or `EZ_LINE` or `EZ_POINT`.

13.16 Display Lists

A display list is a list of GL commands grouped together for later execution. Each display list has a unique integer identifier. A display list is started by calling

```
void EZ_NewList(int lnum, int mode);
```

This function starts a display list. Possible modes are `mode` specifies a mode, either `EZ_COMPILE` or `EZ_COMPILE_AND_EXECUTE`. If mode is `EZ_COMPILE`, the subsequent gl commands are not executed as they are placed in the display list. If mode is `EZ_COMPILE_AND_EXECUTE`, then the commands are executed, and placed in the display list.

A display list is completed by calling

```
void EZ_EndList(void);
```

To execute the commands saved on a display list, use

```
void EZ_CallList(int lnum);
```

To destroy a display list, use

```
void EZ_DestroyList(int lnum);
```

To test whether or not an integer identifies a display list, use

```
int EZ_IsList(int lnum);
```

It is also possible to manipulate multiple lists a time.

`int EZ_GenLists(int range);` This function returns an integer such that the indices $n, n+1, \cdots, n+\mathrm{range}-1$ are not currently used.

```
int EZ_CallLists(int start, int range);
```

This function executes the commands on the lists $\mathrm{start}, \mathrm{start}+1, \mathrm{start}+2, \cdots, \mathrm{start}+\mathrm{range}-1$.

```
int EZ_DeleteLists(int start, int range);
```

This function delete the display lists with indices $\mathrm{start}, \mathrm{start}+1, \mathrm{start}+2, \cdots, \mathrm{start}+\mathrm{range}-1$.

13.17 Miscellaneous Commands

Here are a few more functions for manipulating matrices.

```
void EZ_CopyMatrix(EZ_Matrix des, EZ_Matrix src);
void EZ_CopyTransposeMatrix(EZ_Matrix des, EZ_Matrix src);
void EZ_IdentityMatrix(EZ_Matrix mtx);
void EZ_ZeroMatrix(EZ_Matrix mtx);
void EZ_InvertMatrix4_(EZ_Matrix ans, EZ_Matrix src, int transp);
void EZ_World2Screen(float *xyz, int *x, int *y, float *fz);
void EZ_Screen2World(int x, int y, float *fx, float *fy);
```

Index