Lidskjalv: User Interface Framework − Reference Manual

Mjølner Informatics Report
MIA 94−27
March 2004

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

List of Programs

1 Overview of Lidskjalv

Lidskjalv [1] is an platform independent object−oriented user interface construction framework for constructing user interfaces that are easily portable between the Macintosh window system, the X Window System (based on Motif Widgets), and the Microsoft Windows (Windows NT, Windows 2000, Windows 98, or Windows 95).

This document contains the object−oriented model for Lidskjalv, along with an overview of the interface files for Lidskjalv.

The Lidskjalv model is based on previous experiences in developing object−oriented user interface construction frameworks for the Macintosh window system, and the X Window System (based on Athena and Motif Widgets). It has been an important design criterion to make a model that deals with the construction of portable user interfaces in such a way that the details of the look−and−feel of the Lidskjalv applications will conform to the standardised look−and−feel at the specific platform; i.e. the interface will appear to the user as a genuine Motif interface when running in a Motif environment, and will appear as a genuine Macintosh application when running in a Macintosh environment, etc. However, not all issues can be dealt with without the co−operation from the application programmer. To give an example: if the application programmer decides that the windows in the application should contain individual menubars, there is no way this application can conform with the Macintosh User Interface Guidelines. However, Lidskjalv will allow the application programmer to specify individual window menubars on a Macintosh. However, the application programmer might use other facilities of Lidskjalv to deal with this issue of window−specific menubars, such that the code is purely portable across look−and−feel.

Lidskjalv defines abstractions for all commonly used interface objects, such as window, menubar, menu, button, text fields, figure items, scrolling lists, etc. Each interface object takes care of the interactions related to itself, and it is the responsibility of the entire framework to ensure that the user interactions (such as mouse button presses, key presses, etc.) are taken care of and converted internally into invocations of virtual procedures of the appropriate interface object − that is, no application programmer needs to handle user interaction at the event level of the underlying platform.

Lidskjalv is realised in the form of a class, guienv, whose instance acts as the framework for the user interface, taking care of all platform dependent event handling, etc.

---

[1] Lidskjalv is the name of the Odin's high throne, from which he is able to look into all Worlds and see everything that happens

2 The Lidskjalv Libraries

The Lidskjalv libraries consists of 15 libraries (fragments):

- guienv
- stddialogs
- control
- fields
- figureitems
- scrolllist
- graphmath
- graphics
- styledtext
- guienvactions
- controlsactions
- fieldsactions
- guienvall
- guienvsystemenv

The guienv library defines the basic facilities of the Lidskjav framework (as described above). All Lidskjav programs must include at least the guienv fragment.

The stddialogs library includes a few predefined patterns for simple, standard dialogs, such as fileSelectionDialog.

The control library includes facilities for specifying control interfaceObjects (such as buttons etc.) as described above.

The fields library includes facilities for specifying field interfaceObjects (such as static text fields, editable textfields, etc.)

The figureitems library includes facilities for specifying basic graphics interfaceObjects (such as lines, ovals, etc.)

The scrolllist library includes facilities for specifying scrolling interfaceObjects (such as scrolling lists etc.)

The graphmath library includes facilities for graphics computations, such as definition and manipulation of point, rectangle, etc.)

The graphics library includes facilities for simple graphics, such as line drawings.

The styledtext library includes facilities for working with styled text (i.e text in different fonts and styles). Currently only available on Macintosh platforms.

The guienvactions library defines the actions related to the interfaceObjects, defined in the guienv library (more on actions later).

The controlsactions library defines the actions related to the interfaceObjects, defined in the controls library.

The fieldssactions library defines the actions related to the interfaceObjects, defined in the fields library.

The guienvall library is a very simple library, including all the above libraries. Used for easy access to the entire suite of facilities in guienv.

The guienvsystemenv library is a very simple library that enables concurrency and guienv to work perfectly together.

2.1 Utilities Libraries

Besides these libraries, the Lidskjalv user interface framework includes a number of utility libraries. These libraries are located in the utils sub−directory of the Lidskvalv directory tree. It should be noted, that these libraries contains many very useful facilities, but these libraries are not described in this manual. Please refer to the utilities interface files for details of the very many useful patterns.

2.2 Multimedia Libraries

Lidskjalv contains two important libraries for controlling multimedia devices (CD players and QuickTime movies) on the Windows and Macintosh platforms (Unix platform not supported). These facilities are placed in the multimedia sub−directory of the Lidskjalv directory tree. Please refer to the multimedia interface files for details of these useful patterns.

2.3 OpenGL Libraries

Lidskjalv contains libraries for making OpenGL graphics. These facilities are placed in the openGL sub−directory of the Lidskjalv directory tree. It should, however, be noted that these libraries are not fully developed, and tested, and is included in this release 'as−is'. Please refer to the OpenGL interface files for details of these useful patterns.

3 The guienv Library

Guienv is the most basic library of the Lidskjalv libraries. Guienv implements the most often used elements of a graphical user interface, such as menus, windows, etc. along with a lot of supporting facilities.

The prime elements of guienv is the definition of the menu system and the facilities for defining windows along with facilities for handling the events (such as mouse button press), originating from the user interface (menus, windows, etc.).

The most important classes, defined in guienv are:

- interfaceObject is the root of the hierarchy of interface components in Lidskjalv.
- menuBar is a class, defining the facilities for defining menubars. The global menubar (if any) is accessed through the applicationMenubar attribute. StandardMenubar defines the standard menubar, used by most applications.
- menu is a class, defining the facilities for menus, both menus of menubars, menus associated with buttons, and popup menus.
- window defines the facilities for defining and manipulation windows.

The rest of the attributes defines various other facilities for accessing different other aspects of the window system.

The most important class in Lidskjalv is interfaceObject. It defines the facilities available for controlling all interface components in guienv.

interfaceObject implements the basic facilities for all interaction:

- open and close is invoked when the interfaceObject is opened and closed on the screen.
- enableEventType and disableEventType are used to control which interactions, this interfaceObject is willing to respond to (default is that all event types are enabled).
- Most attributes relate to event handling. Event is the superpattern for all event patterns (such as mouseDovn and refresh). These events are invoked by the underlying system as the result of, e.g. user interactions.

Event handling in guienv is handled through defining virtual further bindings in which the actions to be executed as the result of an event, is specified. Each interfaceObject type defined a series of event virtuels in the eventhandler virtual pattern, and users of these interfaceObjects may then further bind this eventhandler, and in this further binding specify the actions to be executed for the particuler events, defined for that type of interfaceObject.

The basic eventhandler (defined in interfaceObject) defines the following events: onMouseDown, onMouseUp, onKeyDown, onRefresh, onActivate, and onDeactivate. Some of these events carry global information on the state of the keyboard and mouse.

In order to support more dynamic event handling, it is also possible dynamically to attach actions before or after the predefined actions for an event. This is done by specifying an instance of (a subpattern of) action and then either prepend or append it to the already attached actions (using the prependAction or appendAction operations). An action can be retracted again by the deleteAction operation.

Parallel to the event hierarchy is an action hierarchy. An action (e.g. mouseDownAction and refreshAction) can be associated with an event, such that the actions will be invoked either before

or after the events itself. These actions are defined in separate libraries, see chapters 10−12.

In order to control the interfaceObject's sensibility to individuel event types, the operations enableEventType and disableEventType are available.

3.1 Event handling

One of the strengths of the Lidskjalv libraries is the ease with which the event handling is conducted. Essentially, the Lidskjalv libraries takes care of all the details of the event dispatching and handling. The Lidskjalv libraries essentially converts all event occurrences into invocation of special virtual patterns within the appropriate user interface object (e.g. the one below the mouse pointer). In the Lidskjalv documentation, these virtual patterns are often referred to as the event patterns (or simply events). The application programmer only has to further bind these event patterns of the individual user interface objects to specify the actions to be taken in response to user interaction.

3.2 Menu facilities

The menu facilities are centred around the concept of menubars. In guienv, there may be any number of menubars: an application menubar, and a menubar associated with each window. In both cases, these menubars are specified as instances of the menubar pattern:

menubar implements the facilities for defining menubars.

- append, delete and clear are used for manipulating the menus in the menubar.
- appendMenubar, replaceMenubar, and deleteMenubar is used for manipulating a menubar as part of another menubar (e.g. appending all menus in one menubar to another menubar).
- scan facilitates scanning all menus associated with this menubar.

The application menu can be specified by further binding the menubarType virtual. If a standard application menubar is what you want, just further bind to standardMenuBar.

StandardMenubar defines the following important attributes:

- standardFileMenu and standardEditMenu defines the standard file and edit menus. fileMenu, theFileMenu and editMenu, theEditMenu are facilities for specifying different file and edit menus.

The application menubar can be changed and accessed through the applicationMenubar attribute.

The individual menus in a menubar is specified as instances of the pattern menu. menu defines the following important attributes:

- name defines the name of this menu.
- onSelect is invoked when selecting in the menu. Further bind this attribute (defined in the local eventhandler virtual) to specify the actions to be executed when this manu is selected in the menubar.
- menuItem and dynamicMenuItem are used to define the individual menu items, and action is used to specify actions to be associated with dynamic menuItems (see later).
- append, delete, clear and scan are used for manipulating the menu items of this menu.
- popUp is used to pop−up this menu.
- enable and disable is used to control whether the menu is enabled or disabled in the menubar.

The pattern menuItem is the facility for defining the individual items in a menu. Most attributes (name, key, checked, etc.) define the visual appearance of the menu item. Besides the following important attributes are defined:

- onStatus is used to define when the menu item is selectable.
- onSelect is used to specify the actions to be executed when the menu item is selected.
- subMenu is used to attach an entire menu to a menu item (i.e. creating a hierarchical menu).

The separator pattern is a special menuItem that inserts a vertical line in the menu to separate groups of items.

The menuItem pattern assumes that the same actions always must be executed when a menu item is selected. If however, we want a more dynamic behaviour, the pattern dynamicMenuItem must be used. This pattern allows actions to be associated dynamically with the particular menu item. dynamicMenuItem idefines the following important attributes:

- attach and detach is used to associate the actions with this dynamicMenuItem.

The actions to be associated with a dynamicMenuItem must be instances of the menuAction pattern.

3.3 Window facilities

window defines the means for creating windows in applications. It defines the following important attributes:

- The events: onAboutToClose, onActivate, etc. are the facilities for specifying actions to be executed as consequence of user interactions.
- theMenubar and menubarType are the means for associating a menubar with this window. And menubarVisible controls whether the menubar should be visible.
- title is used for specifying the window title.
- position, frame and size are used to control the location of the window.
- floating controls whether this window will float on top of all other windows.
- show and hide is used to control the visibility of the window.
- showModal specifies that this window will be shown as a modal window.
- bringToFront, bringBack and bringBehind is used to control the stacking order of this window.
- target is used to control the keyboard focus within this window.
- windowItem is the central facility for defining the contents of windows. WindowItems can be attached to windows (more details later).
- type controls the appearance of the titlebar of the window: palette, dialog or normal.
- canvas is a subclass of windowItem, implementing a local coordinate system. Just as windows, more than one windowItem may be associated with a canvas. Canvas function as a mean for grouping windowItems within windows (or other canvasses).

windowItem is the central class for specifying the contents of windows (and canvasses). The contents of windows are one instance of canvas, simplifying the design.

- the events:ÊonVisibilityChanged, onFrameChanged, onFatherFrameChanged, onMouseUp, etc. are the means for specifying actions to be executed as the consequence of user interactions (directly or indirectly).
- father is a reference to the canvas (and thereby possibly the window), this windowItem is

associated with.
- frame, position, move and size is used to control the location of the windowItem within the father canvas.
- bindLeft, bindRight, bindBottom and bindTop is used to control the behaviour of this windowItem when the father canvas has changed frame.
- show and hide is used to control whether this windowItem is visible on its father canvas.
- enable and disable is used to control whether this windowItem reacts to mouse and keyboard interactions.
- theCursor and cursorType is used to control the cursor to be displayed within this windowItem.
- drag and resize are facilities to be used for interactive manipulations of this windowItem.

canvas is the only windowItem subclass that allows attachment of other windowItems. In this respect, canvasses resemble windows (that uses a canvas to contain the windowItems of the window).

3.4 Using the guienv Library

The Lidskjalv libraries consists of a number of BETA fragments, where the fragment guienv describes the basic patterns of the library.

The fragments control, fields, scrolllist, figureitems etc. contain additional facilities to those defined in guienv. This chapter and the next chapters gives a thorough description of each of these fragments. Along with the descriptions, examples are given to illustrate the intended use.

The guienv fragment consists of a single pattern guienv where the attributes of a graphical application are described. Patterns like window and menu are described inside guienv.

By specializing guienv, the user can develop a Lidskjalv application using the predefined patterns and objects in guienv.

A Lidskjalv application is invoked by executing an instance of the guienv specialization. Any Lidskjalv application must therefore have the following outline:

```
ORIGIN '~beta/guienv/guienv'
--- program: descriptor ---
guienv
  (# ...
  do ...
  #)
```

In order to reduce the complexity of simple applications, more advanced facilities of Lidskjalv are located in separate fragments. To utilize these facilities, the above outline of a typical Lidskjalv application must be augmented by specifying the additional facilities used. This is done by including the fragment containing the facility. The facilities are located in separate fragments, such as control, fields, scrolllist and figureitems. These fragments can be included as follows:

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/control'
--- program: descriptor ---
guienv
  (# ...
  do ...
  #)
```

where, in this case control, is the name of the desired fragment. If more than one additional facility is needed, the INCLUDE line is simply repeated with the names of the other fragments.

The basic pattern in guienv is interfaceObject, which is the common superpattern for all patterns describing interaction with the user.

When guienv executes inner, a global event handler is started. This event handler loops until the attribute terminate is executed. When an event occurs, the global event handler distributes the event to the interfaceObject in question. It could be a menu or the active window.

3.5 Examples of Use of the guienv Fragment

The demo programs in this manual can be found in the reference demo subdirectory in the guienv directory. The location of the directory in installation−dependent

~beta/guienv/demo/ReferenceDemos

The demo directory contains many more demo programs than is included in this manual. Please inspect the demo directory for other illustrative demo programs. Appendix A contains a short overview of the demos in the demo directory.

This demo program is nearly the simplest possible Lidskjalv program. It opens one window and prints activate (resp. deactivate) each time the window is made active (resp. inactive).

Program 1: simplewindow.bet

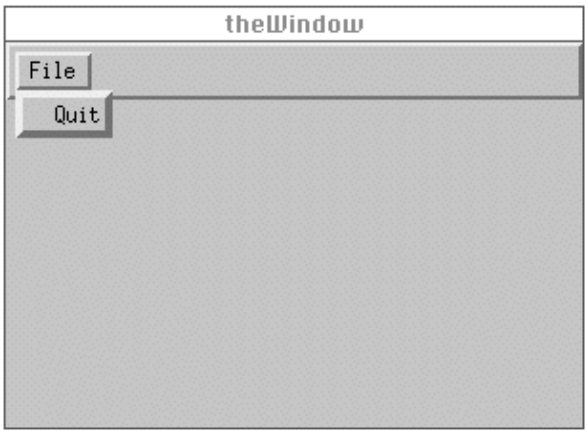
```
ORIGIN '~beta/guienv/guienv';
(* This demo shows how to create a very simple window and it
 * illustrates the activate/deactivate event.
 *)
--- program: descriptor ---
guienv
(# simpleWindow: @window
     (# eventHandler::
          (# onAboutToClose:: (# do terminate #);
             onActivate::
               (# do 'activate' -> putline #);
             onDeactivate::
               (# do 'deactivate' -> putline #)
          #);
        open::
          (# do (400,400) -> size #)
     #)
do simpleWindow.open
#)
```

This demo program illustrates the use of standard menubar where the file menu only has one menuItem. It also illustrates how to further bind noMouseUp, onMouseDown and onKeyDown event patterns.

Program 2: windowWithStdMenubar.bet

```
ORIGIN '~beta/guienv/guienv';
(* This demo shows how to create a simple window with a
 * standardMenubar where the file menu only has one menuitem. It also
 * illustrates how to further bind the onMouseDown, onMouseUp,
 * onKeyDown event patterns.
 *)
--- program: descriptor ---
guienv
(# theWindow: @window
     (# menubarType:: standardMenubar
          (# fileMenu::
               (# quitItem: @menuitem
                    (# open::
                         (# do 'Quit' -> name #);
                      eventHandler::
                         (# onSelect::
                              (# do terminate #)
                         #)
                    #);
                  open::
                    (#
                    do 'File' -> name;
                       quitItem.open; quitItem[] -> append
                    #)
               #)
          #);
        eventHandler::
          (# onAboutToClose:: (# do terminate #);
             onMouseDown::
               (#
               do 'MouseDown: ' -> puttext;
                  buttonState -> putint;
                  (if doubleClick then
                      ' doubleclick' -> puttext
                  if);
                  newLine
               #);
```

```
            onMouseUp::
              (#
              do 'onMouseUp ' -> puttext;
                 buttonState -> putint;
                 (if doubleClick then
                     ' doubleclick' -> puttext
                 if);
                 newLine
              #);
            onKeyDown::
              (#
              do 'onKeyDown: ' -> puttext;
                 ch -> put;
                 newLine
              #)
         #)
     #)
do theWindow.open
#)
```



4 The stddialogs Library

This fragment contains a few standard dialogs (more will be added later):

- noteUser which brings up a simple dialog with a message, and waits for the user to press the OK button.
- alertUser, similar to noteUser.
- fileSelectionDialog brings up a standard file selection dialog.
- fileCreationDialog brings up a standard create/save file dialog.

4.1 Using the stddialogs Library

Remember that in order to utilize this extension to Lidskjalv, the fragment stddialogs must be included as follows:

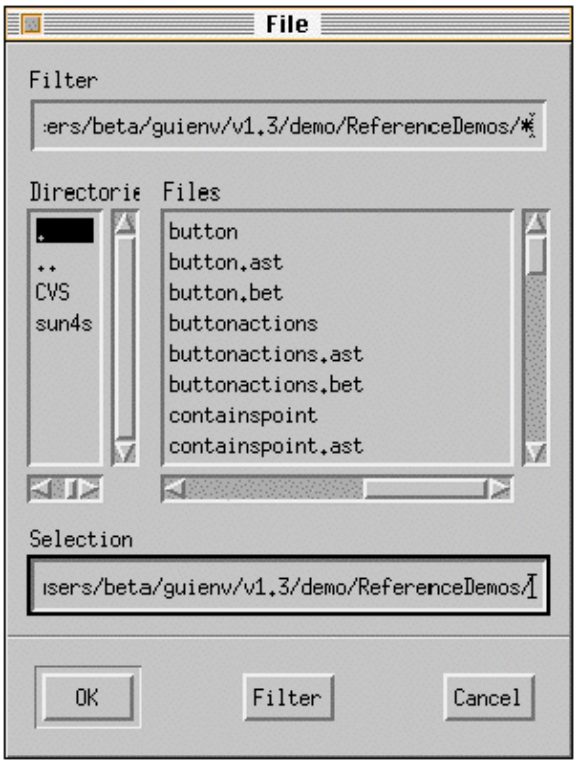
```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/stddialogs'
--- program: descriptor ---
guienv(# ...
      do ...
         ... -> fileSelectionDialog -> ...;
         ...
      #)
```

4.2 Examples of Use of the stddialogs Fragment

This demo program illustrates the use of the standard file selection dialog. The name of the file selected in the dialog is printed on the screen.

Program 3: file.bet

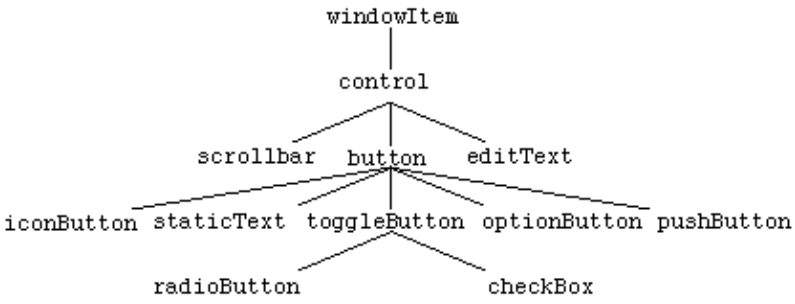
```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/stddialogs';
(* This demo shows how to use the fileSelectionDialog pattern.  The
 * name of the file selected in the dialog is printed on the screen.
 *)
-- program: descriptor --
guienv
(# theWindow: @window
     (# eventHandler::
          (# onAboutToClose:: (# do terminate #);
             onMouseUp::
               (# name: ^text;
               do theWindow[] -> fileSelectionDialog -> name[];
                  (if name[]=NONE then
                      'Selected Cancel' -> putline
                   else
                      name[] -> putline
                  if)
               #)
          #)
     #)
do theWindow.open
#)
```


File
Filter
:ers/beta/guienv/v1.3/demo/ReferenceDemos/*
Directorie
Files
..
CVS
sun4s
button
button.ast
button.bet
buttonactions
buttonactions.ast
buttonactions.bet
containspoint
containspoint.ast
Selection
sers/beta/guienv/v1.3/demo/ReferenceDemos/
OK
Filter
Cancel

5 The controls Library

The controls library contains a series of subpatterns of windowItem, intended primarily to be used in dialog boxes (e.g. buttons, check boxes, etc.).

These subpatterns are called controls, and the inheritance tree of controls in the controls library is:



To illustrate the facilities, we have included the Macintosh graphical elements associated with these classes. Naturally, the graphical elements will appear differently on the Motif and Win32 platforms:

Control pattern name

Image

Description

scrollbar

Used for various scrolling purposes.

staticText

Paper:

Used to specify permanent text in the dialog (usually explanatory text).

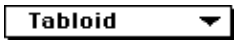
editText

Used to allow the user to enter some text.

pushButton

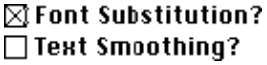
A button is used to specify some actions to be taken.

optionButton



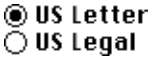
Used to specify a button with associated pop−up menu.

checkBox



A check box is usually used together with other check boxes to present the user with a group of non−exclusive options.

radioButton



A radio box is usually used together with other radio boxes to present the user with a group of exclusive options.

iconButton

An icon is used to show a minor picture in the dialog.

Control is the superclass of all classes in the controls hierarchy. Most facilities of this class are intended for the implementation of the subclasses, and not relevant for most users of Lidskjalv.

Scrollbars are dials which the user can control to specify a value between 0 and some maximum value. The scrollbar pattern has attributes for controlling the scroll step (scrollAmount) and the maximum value of the scrollbar (maxValue). The current value of the scrollbar can be obtained and changed by the value attribute.

The event handling of scrollbar defines several new events, e.g. onThumbMoved which is invoked when the scroll thumb have been moved by the user, and onPageUp which is invoked when the user presses the pageUp area in the scrollbar.

The orientation of the scrollbar is controlled by the vertical attribute (binding vertical to trueObject sets the orientation to vertical). The length of the scrollbar is manipulated through the length attribute.

EditText is a very single line text editor, primarily usable for small amounts of text (such as file names etc.) in dialogs. The text must be in the same text style.

- style is used to specify the font information to be used for the text in this editText.
- contents is a reference to the text of this editText.

Button is the general superpattern for all controls that may act as buttons (i.e. react to menu button clicks), and may have a label associated with them.

Button has attributes for accessing and changing the label (label), and for accessing and changing the text style of the label (style).

- the events: onLabelChanged and onStyleChanged are invoked when the label or style have been changed.
- label is the text displayed in (or immediately along with) the button.
- style is used to define the font to be used when displaying the label.

PushButton is a simple button that reacts to mouse clicks. PushButton does not define additional attributes. The label of a pushButton is shown inside the button.

StaticText is a simple text label, and mostly used for informative text in dialogs and for labeling editText fields. StaticText does not define any additional attributes.

IconButton is a simple button with a icon defining its appearance. The label of an icon button may be shown centered below the icon, The showLabel attribute is used for controlling whether the label should be shown or not.

- the event: onShowLabelChanged is invoked when the label have been changed.
- showLabel is used to control whether the label should be displayed along with the icon.

An optionButton has an associated menu, that pops up when the user clicks at the optionButton. The button text of the optionButton is automatically updated to display the currently selected item in the menu. The currently selected menu item is accessible through the currentItem attribute. The menu connected to this optionButton is an instance of the menu pattern defined in guienv. The menu is connected through the popUpMenu attribute.

- the events: onCurrentItemChanged and onPopupMenuChanged are invoked when the a new item is selected in the associated menu, respectively when the menu is changed.
- popupMenu is used to specify the menu to be popped up from this button.
- currentItem is the last selected menu item.

ToggleButton is the general superpattern for on/off buttons.

- the event: onStateChanged is used to specify the actions to be executed when the state of the toggleButton have been changed.
- state is used to control the state of this toggleButton.

RadioButton is a kind of toggleButton mostly used in a radio button cluster (several radioButtons of which only one can be on at any time − this must however be ensured by the application programmer in the current version). No additional attributes are defined.

CheckBox is mostly used for setting options in e.g. dialog boxes. Is intended to be used in checkbox groups. No additional attributes are defined.

DefaultButton is used for specifying the button to act as the default button (i.e. be activated by a carriage return). DefaultButton takes a reference to the button to be used as default as enter parameter. Currently not implemented on Motif.

5.1 Using the controls Library

Remember that in order to utilize this extension to Lidskjalv, the fragment controls must be included as follows:

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/controls'
--- program: descriptor ---
```

```
guienv(# pb: @pushButton;
         ...
      do ...
         ... -> pb.label;
         ...
      #)
```

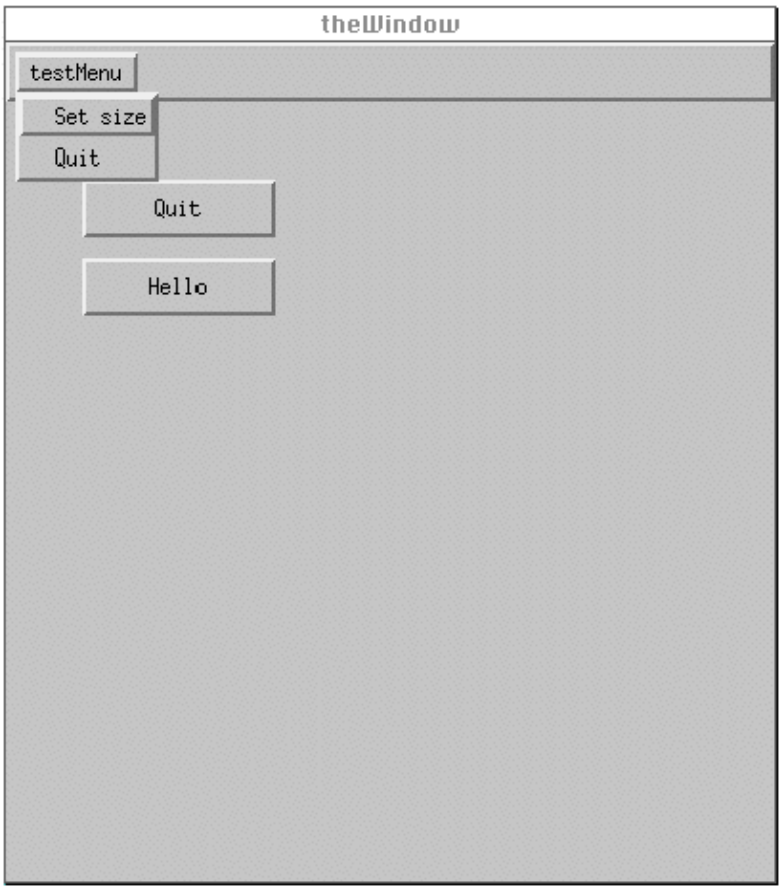
5.2 Examples of Use of the controls Fragment

This example illustrates how to create a window with two pushbuttons, and give a button a new size at runtime.

Program 4: button.bet

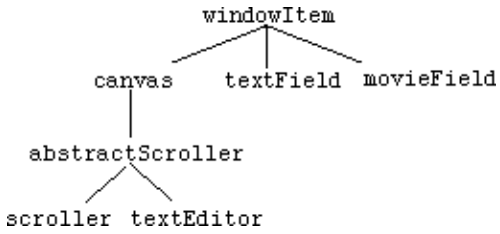
```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/controls';
(* This demo shows how to create a window with two pushButtons, and
 *  how to give a button a new size on runtime.
 *)
--- program: descriptor ---
guienv
(# theWindow: @window
     (# menubarType::
          (# testMenu: @menu
               (# sizeItem: @menuItem
                    (# open::
                         (# do 'Set size' -> name #);
                       eventHandler::
                         (# onSelect::
                              (# do (200,200)->helloButton.size #);
                         #);
                    #);
                  quitItem: @menuItem
                    (# open::
                         (# do 'Quit' -> name #);
                       eventHandler::
                         (# onSelect::
                              (# do terminate #);
                         #);
                    #);
                  open::
                    (#
                    do sizeItem.open; sizeItem[] -> append;
                       quitItem.open; quitItem[] -> append
                    #)
               #);
             open::
               (#
               do testMenu.open; testMenu[] -> append
               #)
          #);
        quitButton: @pushButton
          (# eventHandler::
               (# onMouseUp::
                    (#
                    do 'Good bye, World' -> putline;
                       terminate
                    #)
               #);
             open::
               (#
```

```
               do (40,40) -> position;
                  (100,30) -> size;
                  'Quit' -> label
               #)
         #);
       helloButton: @pushButton
         (# eventHandler::
              (# onLabelChanged::
                   (#
                   do 'helloButton.onLabelChanged' -> putline
                   #);
                 onMouseUp::
                   (#
                   do 'Hello, World − My name is ' -> puttext;
                      label -> putline
                   #)
              #);
            open::
              (#
              do (40,80) -> position;
                 (100,30) -> size;
                 'hello,world' -> putline;
                 'Hello' -> label
              #)
         #);
       eventhandler::
         (# onAboutToClose:: (# do terminate #) #);
       open::
         (#
         do (400,400) -> size;
            quitButton.open;
            helloButton.open;
            contents -> target
         #)
    #)
do theWindow.open
#)
```


theWindow
testMenu
Set size
Quit
Quit
Hello

6 The fields Library

The fields library contains five advanced subpatterns of windowItem. These patterns are used for displaying movies and editable text fields (with and without scrolling facilities), and for scrolling a group of windowItems using scrollbars.



TextField is an alternative to the editText control with extended facilities for cut/copy/paste, selection handling, etc. TextField contains many attributes supporting many different text editing functions: changing text style, deleting and inserting text, etc. Furthermore, textField has an additional event onTextChanged which will be invoked each time the text contents have been changed.

A readonly textField can be implemented simply by further binding the onBeforeChange virtual and here specify false−>allow.

TextField takes care of most interaction with the user and many applications using textField need not do anything else but extract the text contents of the textField instance at the appropriate time.

AbstractScroller is a subpattern of canvas and implements a scrolling facility for any windowItem type (specified through the contentsType virtual attribute). An abstractScroller contains two scrollbars (one vertical and one horizontal), and the windowItem will scroll according to the manipulations of these scrollbars.

TextEditor is a subpattern of abstractScroller (with contentsType further bound to textField) and is an alternative to textField, offering additional scrolling facilities. TextEditor is a full−fledged text editor with full editing and scrolling. No additional attributes are defined.

TextEditor takes care of most interaction with the user and many applications using textEditor need not do anything else but extract the text contents of the textEditor instance at the appropriate time.

Scroller is another subpattern of abstractScroller (with contentsType further bound to canvas) that can scroll all windowItems attached to the canvas.

MovieField is intended for displaying video (not implemented yet).

6.1 Using the fields Library

Remember that in order to utilize this extension to Lidskjalv, the fragment fields must be included as follows:

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/fields'
--- program: descriptor ---
guienv(# te: @textEditor;
         ...
      do ...
         ... -> te.contents.selection;
```

```
        ...
    #)
```

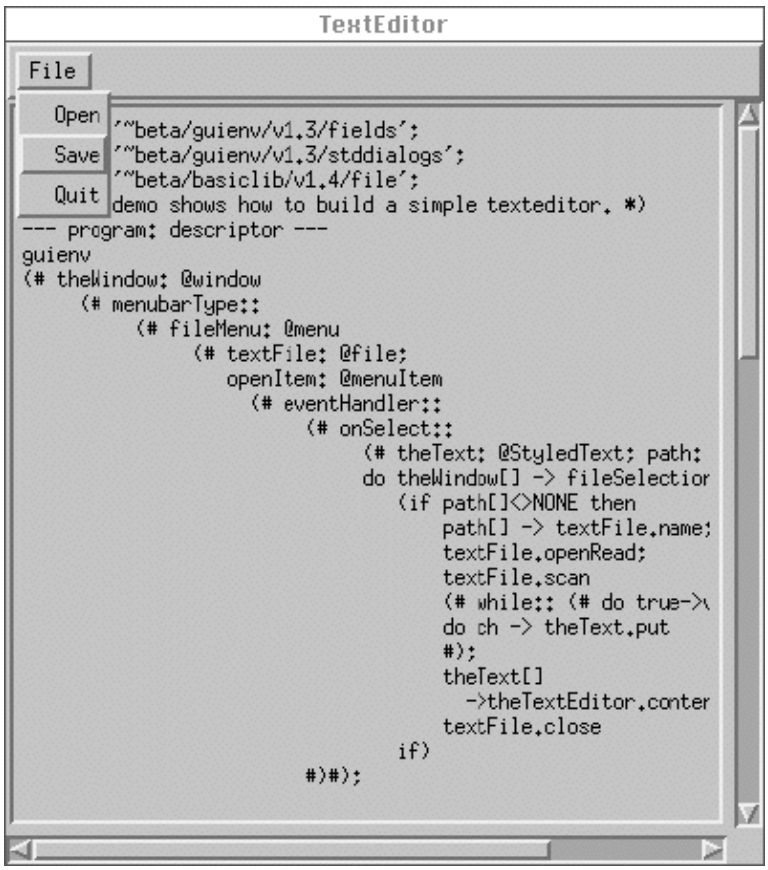
6.2 Examples of Use of the fields Fragment

This demo program illustrates the facilities for constructing standard text editors.

Program 5: texteditor.bet

```
ORIGIN '~beta/guienv/fields';
INCLUDE '~beta/guienv/stddialogs';
INCLUDE '~beta/basiclib/file';
(* This demo shows how to build a simple texteditor. *)
--- program: descriptor ---
guienv
(# theWindow: @window
     (# menubarType::
          (# fileMenu: @menu
               (# textFile: @file;
                  openItem: @menuItem
                    (# eventHandler::
                         (# onSelect::
                              (# theText: @StyledText; path: ^text;
                              do theWindow[] -> fileSelectionDialog -> path[];
                                 (if path[]<>NONE then
                                     path[] -> textFile.name;
                                     textFile.openRead;
                                     textFile.scan
                                     (# while:: (# do true->value #);
                                     do ch -> theText.put
                                     #);
                                     theText[]
                                       ->theTextEditor.contents.contents;
                                     textFile.close
                                 if)
                           #)#);
                        open:: (# do 'Open' -> name #)
                    #);
                  saveItem: @menuItem
                    (# eventHandler::
                         (# onSelect::
                              (# theText: @Text; tempName: ^text;
                              do textFile.name -> tempName[];
                                 (if tempName.length>0 then
                                     textFile.openWrite;
                                     theTextEditor.contents.contents
                                       ->textFile.puttext;
                                     textFile.close
                                 if)
                           #)#);
                        open::
                          (# do 'Save' -> name #)
                    #);
                  quitItem: @menuItem
                    (# eventHandler::
                         (# onSelect:: (# do terminate #) #);
                        open::
                          (# do 'Quit' -> name #)
                    #);
                  open::
                    (#
```

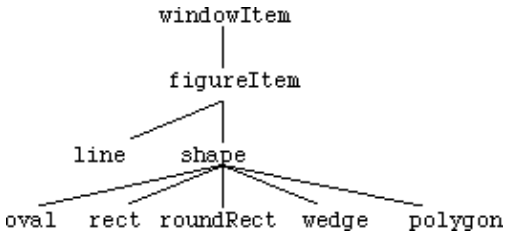
```
                    do 'File' -> name;
                       openItem.open; openItem[] -> append;
                       saveItem.open; saveItem[] -> append;
                       quitItem.open; quitItem[] -> append
                #)#);
              open::
                (# do fileMenu.open; fileMenu[] -> append #)
            #); (* menubarType *)
          eventhandler::
            (# onAboutToClose:: (# do terminate #) #);
          thetextEditor: @textEditor
            (# open::
                 (#
                 do 'TextEditor' -> title;
                    theWindow.size -> size;
                    true -> bindBottom -> bindRight
                 #)
            #);
          open::
            (#
            do 'thetextEditor' -> title;
               (400,400) -> size;
               thetextEditor.open
            #)
       #)
do theWindow.open
#)
```



7 The figureitems Library

The figureitems library is implementing a relative simple vector graphics system, sufficient for many purposes. For more advanced graphics, the Mjølner System contains a full−fledged 2D graphics library, called Bifrost (see [MIA 91−13] and [MIA 91−19]).

The vector graphics patterns are all subpatterns of figureItem. FigureItem is a subpattern of windowItem, and inherits all the event handling etc., allowing figureItems to react to user manipulations.



FigureItem is the general superpattern for all vector graphics patterns, implementing the common attributes of all graphics patterns. All figureItems have the attribute pen. Pen is an object used for drawing the figureItem. Pen has attributes for defining the drawing pattern of pen, for defining the foreground and background color of pen, and for defining the size (rectangle) of pen.

Line is used for drawing straight lines. The start and end attributes are used for controlling the starting (resp. ending) points of the line.

Shape is the general superpattern for all figureItems that can be filled with some paint. It defines one additional attribute, fill. Fill is an object used for modeling the properties of the paint used for filling the figureItem. Fill has attributes for defining the drawing pattern being used for the paint, and for defining the foreground and background color of the paint.

Oval and Rect are very similar in not defining any additional attributes.

RoundRect is similar to rect but with round corners. The roundness of the corners are defined as ovals and controlled through the roundness attribute.

Wedge defines a `piece of cake' and defines startAngle and endAngle for controlling the wedge.

Polygon is a collection of points defining a connected set of straight lines. Polygon defines one additional attribute, points, used to control the points in the polygon.

7.1 Using the figureitems Library

Remember that in order to utilize this extension to Lidskjalv, the fragment figureitems must be included as follows:

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/figureitems'
--- program: descriptor ---
guienv(# l: @line;
         ...
      do ...
         ps1 -> l.start;
         ps2 -> l.end;
         ...
```

```
      #)
```

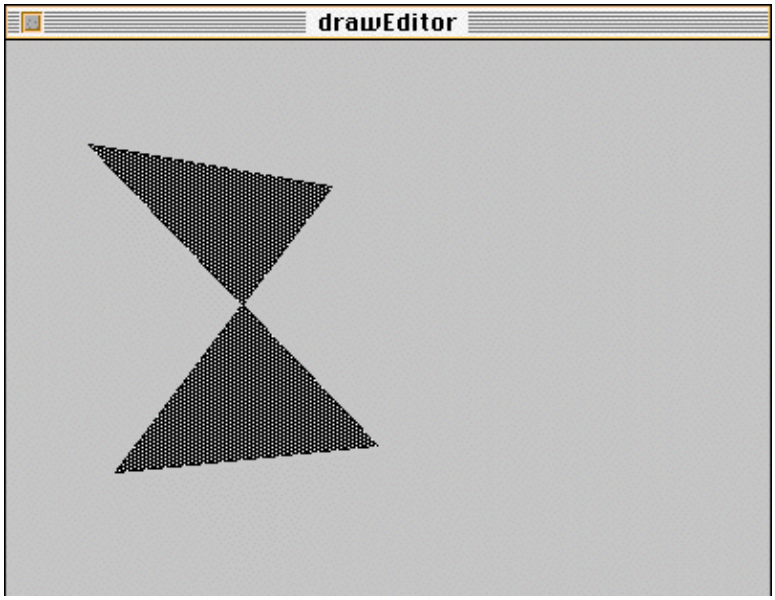
7.2 Examples of Use of the figureitems Fragment

This demo program is a little more elaborate. It is a simple draw editor in which you interactively can draw lines and polygons.

Program 6: draw.bet

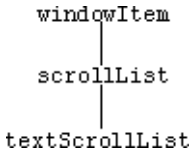
```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/figureitems';
(* This demo gives an example of how you can draw lines and polygons
 * with mouse using the figureitems "line" and "polygon".  Clicking
 * with the left mouse button defines a node in the polygon/line,
 * clicking with the right mouse button ends the definition of the
 * polygon/line and draws it on the screen.
 *)
--- program: descriptor ---
guienv
(# drawEditor: @window
     (# points: @
          (# ps: [16] ^point;
             top: @integer;
             init:
               (#
               do 0 -> top;
                  (for i: ps.range repeat
                       &point[] -> ps[i][]
                  for)
               #);
             clear:
               (# do 0 -> top #);
             add:
               (# p: @point;
                  size: @integer
               enter p
               do top + 1 -> top;
                  (if top > ps.range then
                      ps.range -> size;
                      size -> ps.extend;
                      (for i: size repeat
                           &point[] -> ps[size + i][]
                      for)
                  if);
                  p -> ps[top]
               #);
          exit ps[1:top]
          #);
        polygonEditor: polygon
          (# open::
               (# do patterns.dkgray[] -> fill.tile #);
             eventHandler::
               (# onMouseDown::
                    (#
                    do drag;
                       this(polygonEditor)[] -> father.selection.set
                    #)
               #)
          #);
        lineEditor: line
          (# eventHandler::
```

```
            (# onMouseDown::
                 (#
                 do drag;
                    this(lineEditor)[] -> father.selection.set
                 #)
            #)
        #);
      definingPoly: @boolean;
      eventHandler::
        (# onAboutToClose:: (# do terminate #);
           onMouseDown::
             (# ps: [0] ^point
             do (if definingPoly then
                    (if buttonState
                     //1 then
                        localPosition -> points.add
                     //3 then
                        localPosition -> points.add;
                        points -> ps;
                        (if ps.range > 2 then
                            ps -> createPolygon
                         else
                            ps -> createLine
                        if);
                        false -> definingPoly
                    if);
                 else
                    (if buttonState=1 then
                        true -> definingPoly;
                        points.clear;
                        localPosition -> points.add
                    if)
                if)
             #)
        #);
      createPolygon:
        (# ps: [0] ^point;
           poly: ^polygon
        enter ps
        do &polygonEditor[] -> poly[];
           poly.open;
           ps -> poly.points
        #);
      createLine:
        (# l: ^line;
           ps: [0] ^point
        enter ps
        do &lineEditor[] -> l[];
           l.open;
           ps[1] -> l.start;
           ps[2] -> l.end
        #);
      open::
        (#
        do (400,500) -> size;
           contents -> target;
           points.init
        #)
    #)
do drawEditor.open
#)
```


drawEditor

8 The scrolllists Library

The scrolllists library contains two subpatterns of windowItem which implements scrolling lists:



ScrollList is a general superpattern realizing the basic functionality of displaying and scrolling in a list of equally sized elements (including facilities for managing single and multiple selections of the elements in the scrollList).

ScrollList contains attributes for inserting and deleting elements as well as for managing the selections (inserting, deleting, scanning and testing).

ScrollList is an abstract pattern that cannot be instantiated.

TextScrollList is a minor augmentation of scrollList to make support for scrolling lists of text elements. Adds three additional attributes: setText, getText and style, which allows for accessing and changing the text and the text style of the individual text elements in the scrollList.

8.1 Using the scrolllists Library

Remember that in order to utilize this extension to Lidskjalv, the fragment scrolllists must be included as follows:

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/scrolllists'
--- program: descriptor ---
guienv(# tsl: @textScrollList;
         ...
      do ...
         tsl.selection.first -> ...;
         ...
      #)
```

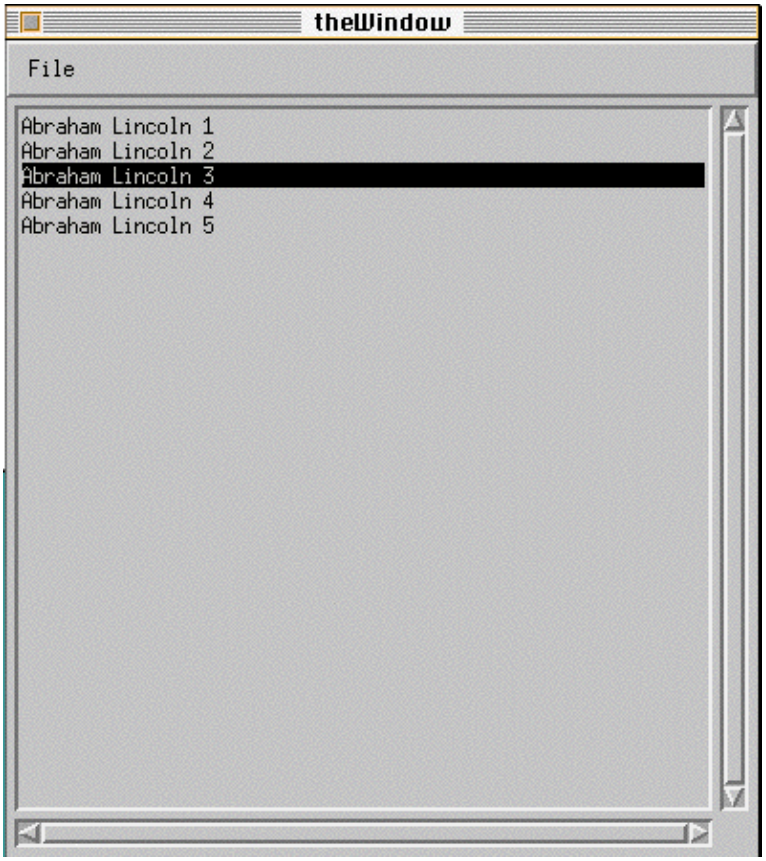
8.2 Examples of Use of the scrolllists Fragment

This example illustrates how to use the textscrolllist windowItem.

Program 7: textscrolllist.bet

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/scrolllists';
(* This demo gives a simple example of how to use the textScrollList
 * windowitem.
 *)
--- program: descriptor ---
guienv
(# theWindow: @window
     (# menubarType::
          (# fileMenu: @menu
```

```
            (# quitItem: @menuItem
                 (# eventHandler::
                      (# onSelect::
                           (# do terminate #);
                      #);
                    open::
                      (# do 'Quit' -> name #)
                 #);
               open::
                 (#
                 do 'File' -> name;
                    quitItem.open; quitItem[] -> append
                 #)
            #);
          open::
            (#
            do fileMenu.open; fileMenu[] -> append
            #)
      #);
     eventhandler::
      (# onAboutToClose:: (# do terminate #) #);
     theTextScrollList: @textScrollList
      (# eventHandler::
           (# onMouseDown::
                (#
                do (if doubleClick then
                       selection.first -> gettext -> putline
                   if)
                #)
           #);
          open::
            (# v: @point;
               noOfItems: @integer
            do (5,5) -> position;
               theWindow.size -> v;
               (10,10) -> v.subtract;
               v -> size;
               true -> bindBottom;
               true -> bindRight;
               5 -> append;
               theTextScrollList.numberOfItems -> noOfItems;
               (noOfItems-4,'Abraham Lincoln 1') -> settext;
               (noOfItems-3,'Abraham Lincoln 2') -> settext;
               (noOfItems-2,'Abraham Lincoln 3') -> settext;
               (noOfItems-1,'Abraham Lincoln 4') -> settext;
               (noOfItems,'Abraham Lincoln 5') -> settext
            #)
      #);
     open::
      (#
      do (400,400) -> size;
         theTextScrollList.open;
         theTextScrollList[] -> target
      #)
  #)
do theWindow.open
#)
```


theWindow
File
Abraham Lincoln 1
Abraham Lincoln 2
Abraham Lincoln 3
Abraham Lincoln 4
Abraham Lincoln 5

9 The graphmath Library

The graphmath library defines patterns for making graphical computations: point, rectangle, matrix, region, etc. Each of these patterns defines several operations.

- point is used for making calculatiuons on 2D positions.
- rectangle is used for making calculations with 2D rectangles
- matrix is used for making calculations with coordinat system transformations.
- region is used for making calculations with 2D regions (i.e. areas in 2D, bounded by a polygon.) Please note that this pattern is not implemented yet.

9.1 Using the graphmath Library

Remember that in order to utilize this extension to Lidskjalv, the fragment graphmath must be included as follows:

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/graphmath'
--- program: descriptor ---
guienv(# p1, p2: @point;
         ...
      do ...
         p1 -> p2.inset -> ...;
         ...
      #)
```

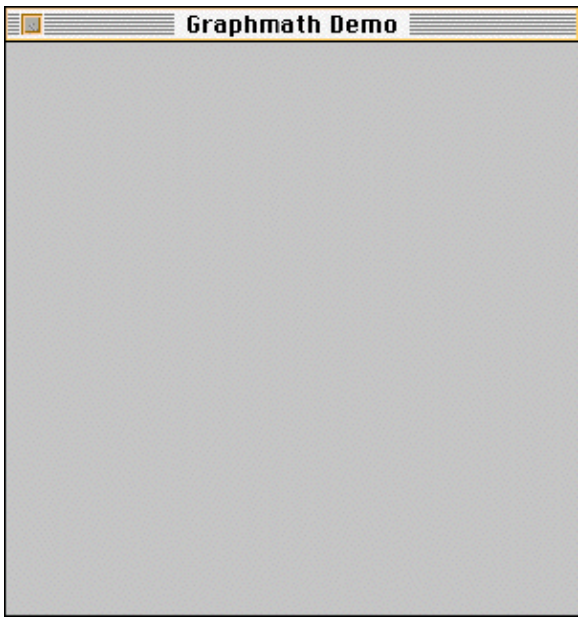
9.2 Examples of Use of the graphmath Fragment

This example gives an example of how to use the containsPoint operation of a rectangle.

Program 8: containspoint.bet

```
ORIGIN '~beta/guienv/guienv';
(* This demo gives an example of how to use the containsPoint
 * operation of a rectangle.
 *)
--- program: descriptor ---
guienv
(# theWindow: @window
     (# hitZone: @rectangle;
        open::
          (#
          do 'Graphmath Demo' -> title;
             (40,40) -> position;
             (300,300) -> size;
             ((0,0), (30,30)) -> hitZone
          #);
        eventHandler::
          (# onAboutToClose:: (# do terminate #);
             onMouseDown::
               (#
               do (if localPosition -> hitZone.containsPoint then
                      'Mouse down in Hit Zone.' -> screen.putline
                  if)
               #)
          #)
```

```
    #)
do theWindow.open
#)
```



10 The graphics Library

The graphics library is defining a simple drawing system (without any event handling facilities). Graphics is intended for lightweight drawings in canvasses, dialogs, etc., where user interaction with the drawings are not important.

Graphics defines a pen for controlling the color, size, etc. of the lines and points to be drawn. Furthermore, graphics defines seleval drawing operations, such as moveTo, drawTo, drawSpot, drawLine, etc.

10.1 Using the graphics Library

Remember that in order to utilize this extension to Lidskjalv, the fragment graphics must be included as follows:

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/graphics'
--- program: descriptor ---
guienv(# pm: @pixmap;
         ...
      do ...
         ... -> pm.read;
         ...
      #)
```

10.2 Examples of Use of the graphics Fragment

This example illustrates reading a pixmap from a file and display it on the screen.

Program 9: drawbitmap.bet

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/graphics';
(* This demo is an example showing how read a bitmap from a file, and
 * using the graphics pattern to draw the bitmap on the screen.
 *)
--- program: descriptor ---
guienv
(# pm: @pixmap;
   theWindow: @window
     (# eventhandler::
          (# onAboutToClose:: (# do terminate #) #);
        obj: @windowItem
          (# open::
               (#
               do (50,50) -> position;
                  (pm.width,pm.height) -> size
               #);
             eventHandler::
               (# onRefresh::
                    (#
                    do graphics
                       (#
                       do (pm[], (0, 0), (0, 0), pm.width,pm.height)
                            -> drawRaster
                       #)
```

```
                    #);
                  onMouseDown::
                    (# do drag #)
                #)
            #);
          open::
            (#
            do (400,400) -> size;
               obj.open
            #)
      #)
do 'picture' -> pm.read;
   (0xffff,0xffff,0xffff)->pm.transparentcolor;
   theWindow.open
#)
```

11 The styledtext Library

The styledtext library is a very small library, implementing the interface to styled text (text in multiple fonts etc.). It is not intended for regular Lidskjalv users, since these facilities are more easily available through other patterns in Lidskjalv (e.g the text editors).

The styledtext library is only available on Macintosh versions. Future releases of Lidskjalv will include some styledtext library on all platforms, however, possibly with a different definition.

11.1 Using the styledtext Library

Remember that in order to utilize this extension to Lidskjalv, the fragment styledtext must be included as follows:

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/styledtext'
--- program: descriptor ---
guienv(# st: @styledText;
         ...
      do ...
         ... -> st.put;
         ...
      #)
```

11.2 Examples of Use of the styledtext Fragment

No demoes, since this extension is not implemented yet.

12 The guienvactions Library

As described in chapter 1, the event handling facilities in guienv includes facilities for attaching actions before and after the predefined actions of an interfaceObject. These actions are created as instances of the action pattern. However, in order to be able to access the informations related to the specific event, specialized action patterns are defined for each event type. These actions are defined in this library (and those described in chapter 12 and 13).

12.1 Using the guienvactions Library

Remember that in order to utilize this extension to Lidskjalv, the fragment guienvactions must be included as follows:

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/guienvactions'
--- program: descriptor ---
guienv(# beforeKeyDown: @keyDownAction
          (# do 'Hello' -> putText #);
        ...
     do ...
        beforeKeyDown[] -> prependAction;
        ...
     #)
```

12.2 Examples of Use of the guienvactions Fragment

12.3 Using keyboard actions

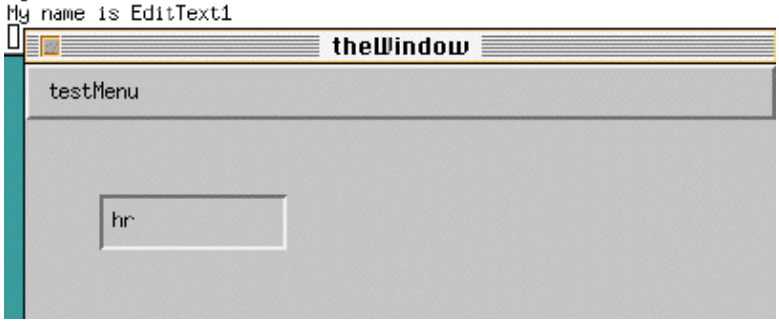
This demo program illustrates the action facilities by attaching actions to be executed before and after onKeyDown events in a edittext control.

Program 10: keyboardactions.bet

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/controls';
INCLUDE '~beta/guienv/guienvactions';
(* This demo shows how to prepend/append actions to keyDown events in
 * an editText control.
 *)
--- program: descriptor ---
guienv
(# theWindow: @window
     (# menubarType::
          (# testMenu: @menu
               (# quitItem: @menuItem
                    (# open::
                         (# do 'Quit' -> name #);
                       eventHandler::
                         (# onSelect::
                              (# do terminate #)
                         #)
                    #);
                  open::
                    (# do quitItem.open;  quitItem[] -> append #)
               #);
             open::
```

```
               (# do testMenu.open; testMenu[] -> append #)
          #);
        eventhandler::
          (# onAboutToClose:: (# do terminate #) #);
        editText1: @editText
          (# beforeKeyDown: @keyDownAction
               (# do 'My ' -> puttext #);
             afterKeyDown1: @keyDownAction
               (# do 'is ' -> puttext #);
             afterKeyDown2: @keyDownAction
               (# do 'EditText1' -> putline #);
             open::
               (#
               do (40,40) -> position;
                  (100,30) -> size;
                  beforeKeyDown[] -> prependAction;
                  afterKeyDown1[] -> appendAction;
                  afterKeyDown2[] -> appendAction
               #);
             eventHandler::
               (# onKeyDown::
                    (# do 'name ' -> puttext #)
               #)
          #);
        open::
          (#
          do (400,400) -> size;
             editText1.open;
             contents -> target
          #)
     #)
do theWindow.open
#)
```

```
My name is EditText1
My name is EditText1
```



13 The controlactions Library

The controlactions library defines the actions related to the events, related to the interfaceObjects, described in the control library.

13.1 Using the controlactions Library

Remember that in order to utilize this extension to Lidskjalv, the fragment controlactions must be included as follows:

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/controlactions'
--- program: descriptor ---
guienv(# lca: @labelChangedAction;
         ...
      do ...
         lca[] -> appendAction;
         ...
      #)
```

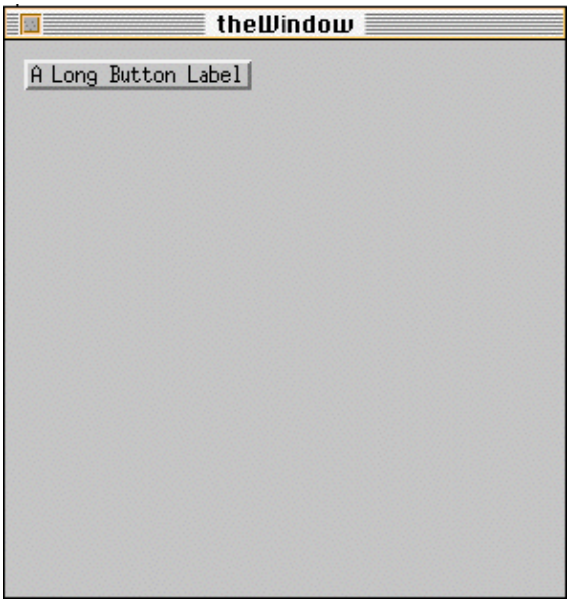
13.2 Examples of Use of the controlactions Fragment

This example illustrates how to use the labelChangedAction to adjust the size of a pushButton to the length of its label.

Program 11: buttonactions.bet

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/controls';
INCLUDE '~beta/guienv/controlsactions';
(* This demo shows how to use the labelChangedAction to adjust the
 * size of a pushButton to the length of its label.  The action is
 * created by remote access to show that actions can be added without
 * specializing the button.
 *)
--- program: descriptor ---
guienv
(# theWindow: @window
     (# theLabelChanged: @aButton.labelChangedAction
          (# theTextStyle: ^textStyle; widthOfLabel: @integer;
             lw,lh: @integer
          do aButton.size -> (lw,lh);
             aButton.style -> theTextStyle[];
             aButton.label -> theTextStyle.widthOfText -> widthOfLabel;
             (if (lw < widthOfLabel + 6) then
                 widthOfLabel + 6 -> lw
             if);
             (if (lh < theTextStyle.lineHeight + 2) then
                 theTextStyle.lineHeight + 2 -> lh
             if);
             (lw,lh) -> aButton.size
          #);
        aButton: @pushButton;
        eventhandler::
          (# onAboutToClose:: (# do terminate #) #);
        open::
          (#
```

```
         do (40,40) -> position;
            (300,300) -> size;
            aButton.open;
            (10,10) -> aButton.position;
            (50,16) -> aButton.size;
            'Button1' -> aButton.label;
            theLabelChanged[] -> aButton.appendAction;
            'A Long Button Label' -> aButton.label
         #)
     #)
do theWindow.open
#)
```



14 The fieldsactions Library

The fieldsactions library defines the actions related to the events, related to the interfaceObjects, described in the fields library.

14.1 Using the fieldsactions Library

Remember that in order to utilize this extension to Lidskjalv, the fragment fieldsactions must be included as follows:

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/fieldsactions'
--- program: descriptor ---
guienv(# bca: @beforeChangeAction;
         ...
      do ...
         bca[] -> appendAction;
         ...
      #)
```

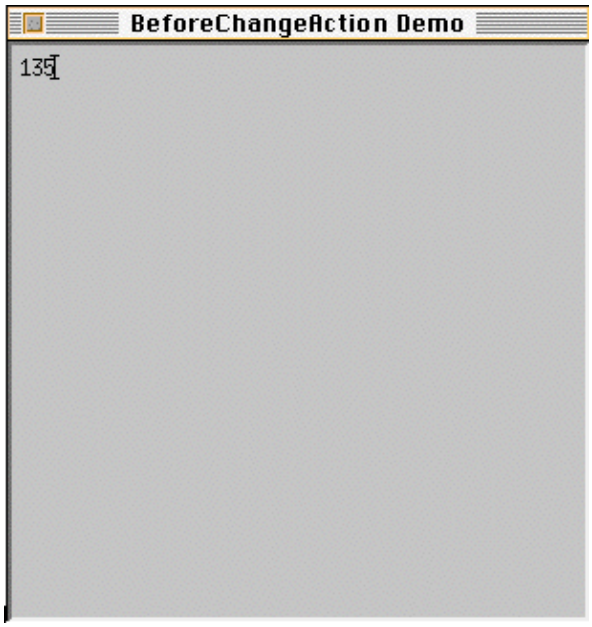
14.2 Examples of Use of the fieldsactions Fragment

This example illustrates how a beforeChangeAction can be udes to 'eat' every second keystroke in a textField.

Program 12: textfieldactions.bet

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/fields';
INCLUDE '~beta/guienv/fieldsactions';
INCLUDE '~beta/guienv/controls';
(* This is a silly demo that shows how a beforeChangeAction can be
 * used to 'eat' every second keystroke in a textField.
 *)
--- program: descriptor ---
guienv
(# allowEdit: @boolean;
   theWindow: @window
     (# theTextField: @textField
          (# aBeforeChangeAction: @beforeChangeAction
               (# do not allowEdit -> allowEdit #);
             eventHandler::
               (# onBeforeChange::
                    (# do allowEdit -> allow #);
               #);
             open::
               (#
               do true -> bindBottom -> bindRight;
                  aBeforeChangeAction[] -> appendAction
               #)
          #);
        eventhandler::
          (# onAboutToClose:: (# do terminate #) #);
        open::
          (#
          do 'BeforeChangeAction Demo' -> title;
             (40,40) -> position;
```

```
            (300,300) -> size;
            true -> allowEdit;
            theTextfield.open;
            (300,300) -> theTextfield.size
        #)
    #)
do theWindow.open
#)
```



15 The guienvall Library

The guienvall library is a utility fragment, including all the Lidskjalv fragments. This library might be useful for the first−time user of Lidskjalv, since he then avoids being concerned with in which library a particular facility is defined. However, please remember, that using the guienvall library will increase your executable, since far to many facilities will be linked into your application.

15.1 Using the guienvall Library

Remember that in order to utilize this extension to Lidskjalv, the fragment guienvall must be included as follows:

```
ORIGIN '~beta/guienv/guienvall';
--- program: descriptor ---
guienv(# pb: @pushButton;
         ...
      do ...
         ... -> pb.label;
         ...
      #)
```

16 The guienvsystemenv Library

The guienvsystemenv library is intended to be used by application, utilizing both Lidskjalv and concurrency. Please refer to the proper manuals for more information on the concurrency facilities.

16.1 Using the guienvsystemenv Library

Remember that in order to utilize this extension to Lidskjalv, the fragment guienvsystemenv must be utilized as follows:

```
ORIGIN 'guienvsystemenv';
--- program: descriptor ---
systemEnv
(# setWindowEnv::< (# do myWindowEnv[] -> theWindowEnv[] #);
   myWindowEnv: @guienv (# ... #);
   ...
#)
```

17 Appendix A: Demo Programs

The demo programs in this manual can be found in the reference demo subdirectory in the guienv directory. The location of the directory in installation−dependent

```
~beta/guienv/demo/ReferenceDemos
```

The demo directory contains many more demo programs than is included in this manual. Please inspect the demo directory for other illustrative demo programs. The following is a short description of the demos in the demo directory:

- simplewindow.bet
  This demo shows how to create a very simple window and it illustrates the activate/deactivate event.
- windowWithStdMenubar.bet
  This demo shows how to create a simple window with a standardMenubar where the file menu only has one menuitem. It also illustrates how to further bind the onMouseDown, onMouseUp and onKeyDown event patterns.
- file.bet
  This demo shows how to use the fileSelectionDialog pattern. The name of the file selected in the dialog is printed on the screen.
- button.bet
  This demo shows how to create a window with two pushButtons, and how to give a button a new size on runtime.
- texteditor.bet
  This demo shows how to build a simple texteditor.
- draw.bet
  This demo gives an example of how you can draw lines and polygons with mouse using the figureitems line and polygon. Clicking with the left mouse button defines a node in the polygon/line, clicking with the right mouse button ends the definition of the polygon/line and draws it on the screen.
- textscrolllist.bet
  This demo gives a simple example of how to use the textScrollList windowitem.
- containspoint.bet
  This demo gives an example of how to use the containsPoint operation of a rectangle.
- drawbitmap.bet
  This demo is an example showing how read a bitmap from a file, and using the graphics pattern to draw the bitmap on the screen.
- iconbutton.bet
  This demo shows how to use the pixmap pattern when creating an iconButton control. Each time you click in the window a new iconButton is created and positioned at the point where you click.
- keyboardactions.bet
  This demo shows how to prepend/append actions to keyDown events in an editText control.
- buttonactions.bet
  This demo shows how to use the labelChangedAction to adjust the size of a pushButton to the length of its label. The action is created by remote access to show that actions can be added without specializing the button.
- textfieldactions.bet
  This is a silly demo that shows how a beforeChangeAction can be used to 'eat' every second keystroke in a textField.

18 Appendix B: Implementation Design

The implementation design for the Lidskjalv framework mostly consists in selecting the widgets and gadgets for implementing the various Lidskjalv components. Below is a table, showing these correspondences for the three target platforms for Lidskjalv: Motif, Macintosh, and Windows (Win32 API).

Lidskjalv Motif Macintosh Win32 API

interfaceObject

widget

superclass for menus, windows, controls, etc.

menuBar

RowColumn created as a menubar

menubar

Menubar

menu

RowColumn created as a menu

menu

Pop−up Menu

window

TopLevelShellWidgetClass

window

window

menuitem

CascadeButton/ PushButton/ ToggleButton

menuitem

menuitem

windowitem

DrawingArea

superclass for controls, texteditors, etc.

ChildWindow (except for figureItems)

canvas

simple specialization of Composite

object defining a local coordinate system with clipping

object defining a local coordinate system with clipping

scroller

ScrolledWindow

a canvas with scrollbars

a canvas with scrollbars

textEditor

ScrolledText

texteditor

MultiLine EDIT Control

control

simple specialization of Core

control (scrollbars, buttons, etc.)

CONTROL

button

simple specialization of Core

superclass for pushbutton, checkbox, etc.

Button Control

pushButton

PushButton

pushbutton

PushButton

iconButton

PushButton showing an image, but with a label

an icon with a name underneath (like the finder icons)

optionButton

OptionButton

popupmenu control

STATIC CONTROL + DROPDOWNLIST

staticText

Label

statictext item

Specialized window item

toggleButton

abstract class

superclass for checkbox and radiobutton

superclass for checkbox and radiobutton

radioButton

ToggleButton with indicatorType = ONE_OF_MANY

radiobutton

Button Control with BS_RADIOBUTTON style

checkBox

ToggleButton with indicatorType = MANY_OF_MANY

checkbox

Button Control with BS_CHECKBOX style

scrollbar

Scrollbar

scrollbar

SCROLLBAR Control

editText

Text without scrollbars configured as a singleline field

edittext item (used for dialogs)

SingleLine EDIT Control

textField

Text without scrollbars configured as a multiline field

multiline textfield with no scrollbars

MultiLine EDIT Control

scrollList

abstract class

superclass for scrolllists

LISTBOX Control

textScrollList

List

scrolllist with one scrollbar, like in the standard file dialog

LISTBOX Control

in interfaceObject:

event

callback

callback functions called when different events occurs

callback functions called when different events occurs

action

eventHandler

callback functions called when different events occurs

callback functions called when different events occurs

open

init (XtCreateWiget/ XtManageChild)

creating and displaying the object

creating and displaying the object

close

destroy (XtDestroyWidget)

removing the object from the screen

removing the object from the screen

mouseDown

buttonPress

mouseDown

WM_LBUTTOMDOWN, WM_MBUTTOMDOWN, WM_RBUTTOMDOWN

mouseUp

buttonRelease

mouseUp

WM_LBUTTOMUP, WM_MBUTTOMUP, WM_RBUTTOMUP

in menu:

name

The LabelString of the CascadeButton the menu is connected to

The name of the menu, as it appears in the menubar

The name of the menu, as it appears in the menubar

in window:

refresh

exposure

update event

WM_PAINT

target

keyBoardFocus

the object that handles the keydown events

the object that handles the keydown events

showModal

the window shown as SYSTEM_MODAL

the window used as a modal dialog

the window used as an application modal dialog

in windowitem:

position

x,y

the topleft corner of the objects bounding box

x,y

size

width,height

the width and height of the bounding box

nWidth,nHeight

bindLeft, etc.

geometry constraints in Form

resize constraints

resize constraints

show

map

show

SW_SHOW

hide

unmap

hide

SW_HIDE

in scrollbar:

scrollAmount

increment

how much the scrollbar scrolls when the arrows are pressed

how much the scrollbar scrolls when the arrows are pressed

pageScrollAmount

pageIncrement

how much the scrollbar scrolls when the page area are clicked

how much the scrollbar scrolls when the page area are clicked

in button descendants:

label

labelString

name

Text of Control

textStyle

fontList

font,face,size of the name

font,face,size of the TEXT Control

in toggleButton descendants:

state

set

state

check state

19 References

[Jones 89]
Oliver Jones: Introduction to the X Window System, Prentice Hall, 1989, ISBN 0−13−499997−5

[Nye & O'Reilly 90a]
Adrian Nye & Tim O'Reilly:
X Toolkit Intrinsics Programming Manual,
Volume Four of `The X Window System Series',
O'Reilly & Associates Inc, 1990, ISBN 0−937175−34−X

[Nye & O'Reilly 90b]
Adrian Nye & Tim O'Reilly:
X Toolkit Intrinsics Programming Manual, OSF/Motif 1.1 Edition,
Volume Four (Motif) of `The X Window System Series',
O'Reilly & Associates Inc, 1990, ISBN 0−937175−62−5

[Poskanzer]
Jef Poskanzer:
Portable BitMap, GrayMap, and PixMap,
UNIX Manual Pages.

[Young 90]
Douglas A. Young:
The X Window System. Programming and Applications with Xt,
OSF/Motif Edition,
Prentice Hall, 1990, ISBN 0−13−497074−8

20.1 Controls Interface

```
ORIGIN 'guienv';
LIB_DEF 'guienvcontrols' '../lib';
BODY 'private/controlsbody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
-- windowLib: Attributes --
control: windowitem
(* a control is a graphical object in the window that the user can
 * use to perform actions. All user interaction with the control
 * that can result in an action should give some kind of visual
 * feedback.
 *)
  (#
     <<SLOT controlLib:Attributes>>;
     open::< (#
          create::<
            (#
            do ...
            #);

       do ...
       #);
     eventHandler::<
       (#
          onEnabledChanged::<
            (#
            ...
            #);

       #);
     close::<  (#  do ... #);
     private:
       @...;

  #)
  (* control *)
  ;
scrollbar: control
(* A scrollbar controls the scrolling of a textfield or picture
 * etc.
 *)
  (#
     <<SLOT scrollbarLib:Attributes>>;
     eventhandler::<
       (#
          thumbMoved: event
          (* is called whenever a user has moved the thumb of
           * THIS(scrollbar)
           *)
            (# amount: @integer;
            enter amount
            do ...;
            #);
          onThumbMoved:< thumbMoved;
          pageDown: event
          (* called when the user clicks in the page down area *)
            (#  do ...;  #);
          onPageDown:< pageDown;
          pageUp: event
```

```
      (* called when the user clicks in the page up area *)
        (#  do ... #);
      onPageUp:< pageUp;
      buttonDown: event
      (* called when the user clicks at the down button *)
        (#  do ... #);
      onButtonDown:<
       buttonDown;
      buttonUp: event (* called when the user clicks at the up button *)
        (#  do ... #);
      onButtonUp:< buttonUp;
      pageScrollAmountChanged:
       event (* called when the scrollamount is changed *)
        (#  do INNER ;  #);
      onPageScrollAmountChanged:< pageScrollAmountChanged;
      scrollAmountChanged: event
      (* called when the scrollamount is changed *)
        (#  do INNER ;  #);
      onScrollAmountChanged:< scrollAmountChanged;
      maxValueChanged: event (* called when the max value is changed *)
        (#  do INNER ;  #);
      onMaxValueChanged:< maxValueChanged;
      valueChanged: event (* called when the value is changed *)
        (#  do INNER ;  #);
      onValueChanged:< valueChanged;
      onFrameChanged::<
        (#  do ... #);
      onRefresh::<
        (#
        do ...
        #);
      onMouseDown::<
        (#  do ... #);
      onActivate::<
        (#
        do ...
        #);
      onDeactivate::<
        (#  do ... #);

  #);
vertical:<
(* Specifies if THIS(scrollbar) is vertical, false is the
 * default value
 *) booleanValue;
scrollAmount:
(* scrollAmount is the amount the scrollbars thumb will move,
 * when the user clicks in the up button or in the down button
 *)
  (# value: @integer
  enter
    (#
    enter value
    do ...
    #)
  exit
    (#
    do ...
    exit value
    #)
  #);
pageScrollAmount:
(* pageScrollAmount is the amount the scrollbars thumb will
 * move, when the user clicks in the page down or page up area
 *)
  (# value: @integer;
```

```
  enter
    (#
    enter value
    do ...
    #)
  exit
    (#
    do ...
    exit value
    #)
  #);
maxValue:
(* this combines setMaxValue and getMaxValue.  Evaluate the
 * enter part to set the maximum value and evaluate the exit
 * part to get the maximum value
 *)
  (# value: @integer
  enter
    (#
    enter value
    do ...
    #)
  exit
    (#
    do ...
    exit value
    #)
  #);
value:
(* evaluate the enter part to set the value and evaluate the
 * exit part to the maximum value.  The thumb of the scrollbar
 * is drawn acording to maxValue and the current value. That
 * is, if maxValue is 100 and value is 50, the thumb will be
 * drawn in the middle of the scrollbar
 *)
  (# pos: @integer;
  enter
    (#  enter pos do ... #)
  exit
    (#
    do ...
    exit pos
    #)
  #);
length:
(* the length is either the height or the width of the frame
 * depending of the orientation
 *)
  (# theLength: @integer;
  enter
    (#
    enter theLength
    do ...
    #)
  exit
    (#
    do ...
    exit theLength
    #)
  #);
open::<
  (# create::<  (#  do ... #)
  do ...
  #);
close::<  (#  do ...;  #);
private:
```

```
      @...;

  #)
  (* scrollbar *)
  ;
button: control
(* this is the abstract superpattern for all button−like controls *)
  (#
     <<SLOT buttonLib:Attributes>>;
     eventhandler::<
       (#
          labelChanged: event
          (* is called whenever the label is changed *)
            (#  do INNER ;  #);
          onLabelChanged:< labelChanged;
          styleChanged: event (* is called whenever the style is changed *)
            (# do INNER #);
          onStyleChanged:< styleChanged;
          onFrameChanged::<
            (#  do ... #);
          onRefresh::<
            (#
            do ...
            #);
          onMouseDown::<
            (#  do ... #);

       #);
     label:
     (* the label is the text displayed in THIS(button).  The event
      * labelChanged is called, when the label is changed
      *)
       (# theLabel: ^text;
       enter
         (#  enter theLabel[] do ... #)
       exit
         (#  do ... exit theLabel[] #)
       #);
     style: (* the text style used for drawing the label  *)
       (# theStyle: ^textStyle
       enter
         (#  enter theStyle[] do ... #)
       exit
         (#  do ... exit theStyle[] #)
       #);
     foregroundColor:
       (# c: @color
       enter c
       do ...
       #);
     open::<
       (#
          create::<
            (#
            do ...
            #)
       do ...
       #);
     close::<
       (#
       do ...
       #);
     private: @...;

  #)
  (* button *)
```

```
  ;
pushButton: button
(* this is a button like the OK and Cancel buttons in dialogs *)
  (#
     <<SLOT pushButtonLib:Attributes>>;
     automaticTarget::< trueObject;
     open::<
       (#
          create::<
            (#
            do ...
            #)
       do ...
       #);
     close::<
       (#
       do ...
       #);
     eventHandler::<
       (#
          onMouseDown::<  (#  ... #);
          onRefresh::<  (#  ... #);
          onHiliteChanged::<
            (#
            ...
            #);

       #);
     private: @...;

  #)
  (* pushButton *)
  ;
staticText: button
(* normally a staticText is used to label editText fields *)
  (#
     <<SLOT staticTextLib:Attributes>>;
     eventhandler::<
       (#
          onRefresh::<
            (#
            do ...
            #);

       #);
     open::<
       (# create::<  (#  do ... #)
       do ...
       #);
     close::<  (#  do ... #);
     private:
       @...;

  #)
  (* staticText *)
  ;
iconButton: button
(* an icon has a label, which is drawn centered just below the
 * image of the icon
 *)
  (#
     <<SLOT iconButtonLib:Attributes>>;
     eventhandler::<
       (#
          showLabelChanged: event
          (* called when showLabel is changed *) (# do INNER #);
```

```
          onShowLabelChanged:< showLabelChanged;
          iconChanged: event (* Called when the icon is changed *)
            (# do INNER #);
          onIconChanged:< iconChanged;
          onRefresh::<
            (#  do ... #);
          onHiliteChanged::<
            (#
            do ...
            #);
          onMouseDown::<  (#  ... #);

        #);
      showLabel:
      (* if true, the label is shown centered under the image of the
       * Icon
       *)
        (# doShow: @boolean
        enter
          (#
          enter doShow
          do ...
          #)
        exit
          (#
          do ...
          exit doShow
          #)
        #);
      icon:
        (# theIcon: ^pixmap;
        enter
          (#
          enter theIcon[]
          do ...
          #)
        exit
          (#
          do ...;
          exit theIcon[]
          #)
        #);
      open::<
        (#
           create::<
             (#
             do ...
             #)
        do ...
        #);
      close::<
        (#
        do ...
        #);
      private: @...;

  #)
  (* iconButton *)
  ;
optionButton: button
(* a optionButton has a menu, which pops up, when the user clicks
 * at the button. A normal way to use a optionButton is to set the
 * label of the button to the current selected item in the menu
 *)
  (#
     <<SLOT optionButtonLib:Attributes>>;
```

```
    eventhandler::<
      (#
         currentItemChanged: event
         (* called when currentItem is changed *) (# do INNER #);
         onCurrentItemChanged:< currentItemChanged;
         popUpMenuChanged: event (* called when popUpMenu is changed *)
           (# do INNER #);
         onPopUpMenuChanged:< popUpMenuChanged;
         onLabelChanged::<
           (#  do ... #);
         onStyleChanged::<
           (#  do ... #);
         onRefresh::
           (#
           ...
           #);

      #);
    currentItem:
    (* the current item is the number of the item, which name is
     * currently shown in the popup box.  You can get a reference
     * to that item by calling: currentItem ->
     * theMenu.getItemByNumber -> theItem[];
     *)
      (# itemNo: @integer
      enter
        (#
        enter itemNo
        do ...
        #)
      exit
        (#
        do ...
        exit itemNo
        #)
      #);
    popupMenu:
    (* evaluate the enter part to set the menu that pops up in
     * THIS(optionButton). And evaluate the exit part to get the
     * menu
     *)
      (# popupMenu: ^menu;
      enter
        (#
        enter popupMenu[]
        do ...
        #)
      exit
        (#
        do ...
        exit PopupMenu[]
        #)
      #);
    open::<
      (#
         create::<  (#  do ... #)
      do ...
      #);
    close::<
      (#
      do ...
      #);
    private: @...;

  #)
  (* optionButton *)
```

```
  ;
toggleButton: button
(* this is the abstract superpattern for all buttons that toggle
 * between two states (on/off buttons)
 *)
  (#
     <<SLOT toggleButtonLib:Attributes>>;
     eventhandler::<
       (#
          stateChanged: event
          (* this event is called whenever the state of
           * THIS(toggleButton) is changed
           *) (# do INNER #);
          onStateChanged:< stateChanged;
          onMouseUp::<  (#  ... #);

       #);
     state:
       (# theState: @boolean;
       enter
         (#
         enter theState
         do ...
         #)
       exit
         (#
         do ...
         exit theState
         #)
       #);
     open::<
       (#
          create::<
            (#
            do ...
            #)
       do ...
       #);
     close::<
       (#
       do ...
       #);
     private: @...;

  #)
  (* toggleButton *)
  ;
radioButton: toggleButton
(* a radioButton is mostly used in a radiobutton cluster, where
 * only one radioButton is set at a time. A radioButton is thus
 * useful to let the user choose among different alternatives
 *)
  (#
     <<SLOT radioButtonLib:Attributes>>;
     open::<
       (#
          create::<
            (#
            do ...
            #)
       do ...
       #);
     close::<
       (#
       do ...
       #);
```

```
     private: @...;

  #)
  (* radioButton *)
  ;
checkBox: toggleButton
(* this is useful for setting options in dialogs *)
  (#
     <<SLOT checkBoxLib:Attributes>>;
     automaticTarget::< trueObject;
     open::<
       (#
          create::<
            (#
            do ...
            #);

       do ...
       #);
     close::<
       (#
       do ...
       #);
     private: @...;

  #)
  (* checkBox *)
  ;
editText: control
(* this is a simple version of textField. Only one textStyle is
 * allowed.  The purpose of this control is to build dialogs
 *)
  (#
     <<SLOT editTextLib:Attributes>>;
     automaticTarget::< trueObject;
     style:
     (* an editText can only have one textStyle.  Evaluate the
      * enter part to set the textStyle.  Evaluate the exit part to
      * get the textStyle
      *)
       (# txStyle: ^textStyle
       enter
         (#
         enter txStyle[]
         do ...
         #)
       exit
         (#
         do ...
         exit txStyle[]
         #)
       #);
     contents:
     (* the contents of an editText is text.  Evaluate the enter
      * part to set the contents, and evaluate the exit part to get
      * the contents
      *)
       (# str: ^text
       enter
         (#  enter str[] do ... #)
       exit
         (#  do ... exit str[] #)
       #);
     eventhandler::<
       (#
          onFrameChanged::<
```

```
            (#  do ... #);
          onKeyDown::<
            (#
            do ...
            #);
          onMouseDown::<
            (#  do ... #);
          onRefresh::<
            (#
            do ...
            #);
          onEnableTarget::<
            (#  do ... #);
          onDisableTarget::<
            (#
            do ...
            #);

        #);
      open::<
      (* the textStyle of THIS(editText) is initially set to the
       * system's textStyle
       *)
        (# create::<  (#  do ... #);
        do ...
        #);
      close::<  (#  do ... #);
      private:
        @...;

  #)
  (* editText *)
  ;
defaultButton:
(* the defaultButton in the window recieves a mouseUp event when
 * the user presses the return−key
 *)
  (# theButton: ^button
  enter
    (#  enter theButton[] do ... #)
  exit
    (#
    do ...
    exit theButton[]
    #)
  #)
```

20.2 Controlsactions Interface

```
ORIGIN 'controls';
INCLUDE 'guienvactions';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
-- scrollbarLib: attributes --
thumbMovedAction: action
  (# eventType::< theEventhandler.thumbMoved
  do INNER
  #);
pageDownAction: action
  (# eventType::< theEventhandler.pageDown
  do INNER
  #);
pageUpAction: action
  (# eventType::< theEventhandler.pageUp
  do INNER
  #);
buttonDownAction: action
  (# eventType::< theEventhandler.buttonDown
  do INNER
  #);
buttonUpAction: action
  (# eventType::< theEventhandler.buttonUp
  do INNER
  #);
pageScrollAmountChangedAction: action
  (# eventType::< theEventhandler.pageScrollAmountChanged
  do INNER
  #);
scrollAmountChangedAction: action
  (# eventType::< theEventhandler.scrollAmountChanged
  do INNER
  #);
maxValueChangedAction: action
  (# eventType::< theEventhandler.maxValueChanged
  do INNER
  #);
valueChangedAction: action
  (# eventType::< theEventhandler.valueChanged
  do INNER
  #);

-- buttonLib: attributes --
labelChangedAction: action
  (# eventType::< theEventhandler.labelChanged
  do INNER;
  #);
styleChangedAction: action
  (# eventType::< theEventhandler.styleChanged
  do INNER
  #);

-- iconButtonLib: attributes --
showLabelChangedAction: action
  (# eventType::< theEventhandler.showLabelChanged;
  do INNER
  #);
iconChangedAction: action
  (# eventType::< theEventhandler.iconChanged
```

```
  do INNER;
  #);

-- optionButtonLib: attributes --
currentItemChangedAction: action
  (# eventType::< theEventhandler.currentItemChanged
  do INNER
  #);
popUpMenuChangedAction: action
  (# eventType::< theEventhandler.popUpMenuChanged;
  do INNER
  #);

-- toggleButtonLib: attributes --
stateChangedAction: action
  (# eventType::< theEventhandler.stateChanged;
  do INNER
  #)
```

20.3 Fields Interface

```
ORIGIN 'guienv';
INCLUDE 'controls'
        'styledtext';
LIB_DEF 'guienvfields' '../lib';
BODY 'private/fieldsbody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
-- windowLib: Attributes --
movie: (#  #) (* ONLY defined to make this fragment compilable *) ;
movieField: windowItem
  (#
     <<SLOT movieFieldLib:Attributes>>;
     contents:
     (* the movie shown in THIS(movieField) *)
       (# theMovie: ^movie
       enter
         (#
         enter theMovie[]
         do ...
         #)
       exit
         (#
         do ...
         exit theMovie[]
         #)
       #);
     scaleToFit:
     (* if true, contents will be scaled to fit in
      * THIS(movieField).  Otherwise, it will be clipped.
      *)
       (# value: @boolean;
       enter
         (#
         enter value
         do ...
         #)
       exit
         (#
         do ...
         exit value
         #)
       #);
     open::< (#
          create::<
            (#
            do ...
            #);

       do ...
       #);
     close::<
       (#
       do ...;

       #);
     private: @...;

  #)
  (* movieField *)
```

```
  ;
textField: windowItem
(* this is a simple field that is used to edit styled text.  There
 * is no scroll functionality, use the textEditor pattern, if
 * scrolling is required. The normal editing commands cut, copy,
 * paste, clear are supported. THIS(textField) has to be the
 * window's target, when editing is performed. this can be obtained
 * by calling THIS(textField)[] −> target... this is automatically
 * done when the user clicks in a textField that isn't the target
 * already
 *)
  (#
     <<SLOT textFieldLib:Attributes>>;
     controlModified: @boolean;
     eventhandler::<
       (#
          textChanged: event
          (* this event is called whenever the text in
           * THIS(textField) is changed
           *) (# do INNER #);
          onTextChanged:< textChanged;
          beforeChange: event
          (* This is called before any change is performed in
           * THIS(textField) If allow is set to false, then change
           * is not performed.  Position indicates where in the
           * textfield, the text is inserted or deleted. Length
           * indicates how many characters is inserted or
           * deleted. If length is negative, then the characters are
           * deleted − otherwise they are inserted.
           *)
            (#
               position,length: @integer;
               allow: @boolean;
               theText: (* The text beeing inserted when lenght > 0 *)
                 (# value: ^text;
                 ...
                 exit value[]
                 #);

            enter (position,length)
            do
               (if controlModified then false−>allow;  else true−>allow if);
               INNER ;
               false−>controlModified;

            exit allow
            #);
          onBeforeChange:< beforeChange;
          onFrameChanged::<
            (#  do ... #);
          onKeyDown::<
            (#
            do ...
            #);
          onMouseDown::<
            (#  do ... #);
          onMouseUp::<
            (#
            do ...
            #);
          onRefresh::<
            (#  do ... #);
          onEnableTarget::<
            (#
            do ...
            #);
```

```
      onDisableTarget::<
        (#  do ... #);

  #);
paste:
(* this method pastes text from the clipboard into
 * THIS(textField) at the current insertion point or replaces
 * the current selection.  The text is styled according to the
 * style information found in the scrap; if there is none, it
 * is given the same style as the first character of the
 * replaced selection (or that of the preceding character if
 * the selection is an insertion point)
 *) (#  do ... #);
copy:
(* the current selection is copied into the clipboard with the
 * associated style information. If the current selection is an
 * insertion point the clipboard is emptied
 *) (#  do ... #);
cut:
(* the current selection is first copied into the clipBoard
 * and then deleted
 *) (#  do ... #);
clear:
(* the current selection is deleted, and the clipboard is not
 * affected.  Calling delete is the same as pressing backspace
 *) (#  do ... #);
contents:
(* the text in THIS(textField) *)
  (# theText: ^styledText;
  enter
    (#
    enter theText[]
    do ...
    #)
  exit
    (#
    do ...
    exit theText[]
    #)
  #);
getChar:
(* returns the character at position (pos) in THIS(textField).
 * The return character (ASCII.cr) and other control characters
 * count
 *)
  (# pos: @integer; ch: @char;
  enter pos
  do ...
  exit ch
  #);
length: integerValue
(* returns the number of characters in THIS(textField) *)
  (#  do ... #);
all:
(* if you want to scan all the text in THIS(textField), use:
 * all -> scanText(#...#)
 *) (#  exit (0,length) #);
scanText:
(* INNER is called for every character from position start to
 * end in THIS(textField). The variable ch is the current
 * character
 *)
  (# start,end: @integer; ch: @char;
  enter (start,end)
  do ...
  #);
```

```
posToPt:
(* Calculates the coordinates of the character number `pos' in
 * the textField.
 *)
  (# pos: @integer; pt: @point;
  enter pos
  do ...;
  exit pt
  #);
ptToPos:
(* Calculates the character that are located at the specified
 * coordinates in the textField.
 *)
  (# pos: @integer; pt: @point;
  enter pt
  do ...;
  exit pos
  #);
selection: @
(* selection is the current range of characters in this
 * (textField) that is selected. Start is the position in the
 * text of the first character of the selection and end is the
 * position of the last.  If the selection is an insertion
 * point, "start" and "end" will be the position of the
 * character just after the carret
 *)
  (#
     start: integerValue
       (#  do ... #);
     end: integerValue
       (#
       do ...
       #);
     contents: (* returns the text selected in THIS(textField) *)
       (# theText: ^text;
       do ...
       exit theText[]
       #);
     scrollIntoView:
     (* scrollIntoView makes sure the Selection is visible,
      * scrolling the textField, if necesary
      *) ...;
     set:
     (* this makes [theStart,theEnd] the new selection *)
       (# theStart,theEnd: @integer;
       enter (theStart,theEnd)
       do ...
       #);
     get:
       (#
       exit (start,end)
       #);

  enter set
  exit get
  #)
  (* selection *)
  ;
defaultStyle:
(* The default style is the style that is used when the
 * textfield has been completely empty and new text is entered.
 *)
  (# style: ^textStyle;
  enter
    (#
    enter style[]
```

```
    do ...
    #)
  exit
    (#
    do ...
    exit style[]
    #)
  #);
isOneStyle:
(* this function returns a textStyle if the range of
 * characters [start,end] has the same style − in which case
 * "theStyle" will be set to that textStyle − Otherwise
 * theStyle will be NONE
 *)
  (# start,end: @integer; theStyle: ^textStyle;
  enter (start,end)
  do ...
  exit theStyle[]
  #);
setOneSize:
(* this makes the range of characters [start,end] have the
 * same size specified by "theSize"
 *)
  (# start,end: @integer; theSize: @integer;
  enter (start,end,theSize)
  do ...
  #);
setOneFont:
(* this makes the range of characters [start,end] have the
 * same font specified by "theFont"
 *)
  (# start,end: @integer; theFont: ^text;
  enter (start,end,theFont[])
  do ...
  #);
setOneFace:
(* this makes the range of characters [start,end] have the
 * same face (textFaces.italic, textFaces.bold etc.) specified
 * by "theFace". If doToggle is true and the face specified
 * exists across the entire selected range, that face is
 * removed (turned off).  Otherwise, all of the selected text
 * is set to include that face
 *)
  (# start,end: @integer; doToggle: @boolean; theFace: @integer;
  enter (start,end,theFace,doToggle)
  do ...
  #);
setOneStyle:
(* this makes the range of characters [start,end] have the
 * same continous style specified by "theStyle"
 *)
  (# start,end: @integer; theStyle: ^textStyle;
  enter (start,end,theStyle[])
  do ...
  #);
scanTextWithStyle:
(* this is a control pattern that calls an INNER for all
 * characters in THIS(textField) with the style "theStyle".
 * The variable "ch" is the current character
 *)
  (# theStyle: ^textStyle; ch: @char;
  enter theStyle[]
  do ...
  #);
margin:
(* use this pattern to set or retrieve the left− and top
```

```
       * margin of the text in THIS(textField).  The left margin is
       * the distance from the left bound of THIS(textField) to the
       * text in THIS(textField).  The top margin is the distance
       * from the upper bound of THIS(textField) to the text in
       * THIS(textField)
       *)
        (# leftMargin,topMargin: @integer;
        enter
          (#
          enter (leftMargin,topMargin)
          do ...
          #)
        exit
          (#
          do ...
          exit (leftMargin,topMargin)
          #)
        #);
      insert:
      (* insert takes the specified text and inserts it just before
       * the selection range in THIS(textField). Insert doesn't
       * affect either the current selection range or the clipboard
       *)
        (# theText: ^text;
        enter theText[]
        do ...
        #);
      delete:
      (* deletes the characters in the current selection range *)
        (#  do ... #);
      open::<
        (#
           create::<
             (#
             do ...
             #);

        do ...
        #);
      close::<
        (#
        do ...
        #);
      private: @...;

  #)
  (* textField *)
  ;
abstractScroller: canvas
(* this is an abstract superpattern for objects with two
 * scrollbars.  The abstractScroller consist of a canvas containing
 * the virtual definition of contents that models the object that is
 * scrolled and the two scrollbars.  It also defines the virtual
 * procedure patterns scroll and adjustscrolling
 *)
  (#
     <<SLOT abstractScrollerLib:Attributes>>;
     contentsType:<
     (* this describes the object that is scrolled *) windowItem;
     contents: @contentsType;
     scroll:<
     (* this is a superpattern for scrolling functionality of
      * THIS(abstractScroller).  The contents are scrolled "dh"
      * pixels to the right and "dv" pixels down
      *)
       (# dh,dv: @integer
```

```
      enter (dh,dv)
      do ...;
      #);
    open::<
      (#
         create::<
           (#
           do ...
           #);

      do ...
      #);
    close::<
      (#
      do ...
      #);
    private: @...;

  #)
  (* abstractScroller *)
  ;
textEditor: abstractScroller
(* this models a texteditor, that is a textfield with two
 * scrollbars
 *)
  (#
     <<SLOT textEditorLib:Attributes>>;
     contentsType::< textField;
     scroll::<
       (#
       do ...
       #);
     open::<
       (#
          create::<  (#  do ... #);
       do ...
       #);
     close::<
       (#
       do ...
       #);
     private: @...;

  #)
  (* textEditor *)
  ;
scroller: abstractScroller
(* this is a general scroller, which can scroll an entire canvas *)
  (#
     <<SLOT scrollerLib:Attributes>>;
     contentsType::< canvas;
     scroll::<
       (#
       do ...
       #);
     open::<
       (# create::<  (#  do ... #);
       do ...
       #);
     close::<  (#  do ...;  #);
     eventhandler::<
       (#
          onFrameChanged::<
            (#
            do ...;
```

```
            #);

      #);
   private: @...;

#)
```

20.4 Fieldsactions Interface

```
ORIGIN 'fields';
INCLUDE 'guienvactions';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
-- textfieldLib: attributes --
textChangedAction: action
  (# eventType::< theEventhandler.textChanged
  do INNER;
  #);
beforeChangeAction: action
  (# eventType::< theEventhandler.beforeChange;
  do INNER;
  #)
```

20.5 Figureitems Interface

```
ORIGIN 'guienv';
LIB_DEF 'guienvfigureitems' '../lib';
BODY 'private/figureitemsbody'
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
-- windowLib: attributes --
figureItem: windowitem
  (* superclass for all vector graphics *)
  (# <<SLOT figureItemLib: attributes>>;
     pen: @
       (* this item models the properties of the pen used to draw the
        * outline of THIS(figureItem)
        *)
       (# foregroundColor:
            (* sets the foreground color of the pen used to draw
             * THIS(figureItem)
             *)
            (# theColor: @color;
            enter (# enter theColor do ... #)
            exit (# do ... exit theColor #)
            #);
          backgroundColor:
            (* sets the background color of the pen used to draw
             * THIS(figureItem)
             *)
            (# theColor: @color;
            enter (# enter theColor do ... #)
            exit (# do ... exit theColor #)
            #);
          stipple:
            (* The pattern used for stippling when drawing with the
             * pen
             *)
            (# p: ^pixmap;
            enter (# enter p[] do ... #)
            exit (# do ... exit p[] #)
            #);
          size:
            (* sets the size of the pen used to draw THIS(figureItem)
             *)
            (# value: @integer;
            enter (# enter value do ... #)
            exit (# do ... exit value #)
            #);
       #) (* pen *);
     open::<
       (* The initialy pen characteristics of THIS(figureItem) are
        *   stiple          = patterns.black
        *   foreGroundColor = colors.black
        *   backGroundColor = colors.white
        *   size      = 1
        *)
       (# create::< (# do ... #);
       do ...
       #);
     eventhandler::<
       (# onRefresh::<
            (# do ...; #);
       #);
```

```
      private: @...;
  #) (* figureItem *);
line: figureItem
  (* straight line defined by a startPt and a endPt *)
  (# <<SLOT lineLib: attributes>>;
      start:
        (# theStart: @point;
        enter (# enter theStart do ... #)
        exit (# do ... exit theStart #)
        #);
      end:
        (# theEnd: @point;
        enter (# enter theEnd do ... #)
        exit (# do ... exit theEnd #)
        #);
      open::<
        (# do ... #);
      eventhandler::<
        (# onRefresh::<
             (# do ... #);
           onFrameChanged::<
             (# do ... #);
           onHiliteChanged::<
             (#  do ... #);
        #);
      private: @...;
  #) (* line *);
shape: figureItem
  (* figures that can be filled *)
  (# <<SLOT shapeLib: attributes>>;
      fill: @
        (* This item models the properties of the fill of THIS(shape)
         *)
        (# tile:
             (* Sets the tile raster used to fill THIS(figureItem) *)
             (# p: ^pixmap;
             enter (# enter p[] do ... #)
             exit (# do ... exit p[] #)
             #);
           foregroundColor:
             (* Sets the foreground color of the pen used to draw
              * THIS(figureItem).
              *)
             (# theColor: @color;
             enter (# enter theColor do ... #)
             exit (# do ... exit theColor #)
             #);
           backgroundColor:
             (* Sets the background color of the pen used to draw
              * THIS(figureItem).
              *)
             (# theColor: @color;
             enter (# enter theColor do ... #)
             exit (# do ... exit theColor #)
             #);
        #);
      open::<
        (* The fill of THIS(shape) is initially:
         *   colorForeground = black
         *   colorBackground = white
         *)
        (# do ... #);
      eventhandler::<
        (# onRefresh::<
             (# do ... #);
           onHiliteChanged::<
```

```
            (#  do ... #);
        #);
      private: @...;
  #) (* shape *);
oval: shape
  (* the oval is defined by a rectangle *)
  (# <<SLOT ovalLib: attributes>>;
     open::< (# do ... #);
     eventhandler::<
       (# onRefresh::<
            (# do ... #);
       #);
  #) (* oval *);
rect: shape
  (# <<SLOT rectLib: attributes>>;
     open::< (# do ... #);
     eventhandler::<
       (# onRefresh::<
            (# do ... #);
       #);
  #) (* rect *);
roundRect: shape
  (* rectangular shape with rounded corners *)
  (# <<SLOT roundRectLib: attributes>>;
     open::<
       (# do ... #);
     roundness:
       (* the corner roundness is specified by means of an Oval *)
       (# theOvalHeight,theOvalWidth: @integer;
       enter (# enter (theOvalHeight,theOvalWidth) do ... #)
       exit (# do ... exit (theOvalHeight,theOvalWidth) #)
       #);
     eventhandler::<
       (# onRefresh::<
            (# do ... #);
       #);
     private: @...;
  #) (* roundRect *);
wedge: shape
  (* a piece of cake *)
  (# <<SLOT wedgeLib: attributes>>;
     open::<
       (# do ... #);
     startAngle:
       (* evaluate the enter part to set the angle, where THIS(wedge)
        * starts.  Evaluate the exit part to get it
        *)
       (# angle: @integer;
       enter (# enter angle do ... #)
       exit (# do ... exit angle #)
       #);
     endAngle:
       (* evaluate the enter part to set the angle, where THIS(wedge)
        * ends.  Evaluate the exit part to get it
        *)
       (# angle: @integer;
       enter (# enter angle do ... #)
       exit (# do ... exit angle #)
       #);
     eventhandler::<
       (# onRefresh::<
            (# do ... #);
       #);
     private: @...;
  #) (* wedge *);
polygon: shape
```

```
(# <<SLOT polygonLib: attributes>>;
   points:
     (* set or get the points that represents THIS(polygon).  There
      * must be at least 3 points
      *)
     (# thePoints: [3] ^point;
     enter (# enter thePoints do ... #)
     exit (# do ... exit thePoints #)
     #);
   open::<
     (#
     do ...;
     #);
   eventhandler::<
     (# onRefresh::<
          (# do ... #);
        onFrameChanged::<
          (# do ... #);
     #);
   private: @...;
#)
```

20.6 Graphics Interface

```
ORIGIN 'guienv';
LIB_ITEM 'guienv';
BODY 'private/graphicsbody'
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
-- windowitemLib: attributes --
graphics:
  (* The graphics pattern is intended to implement a basic drawing
   * facility for canvases.
   *
   * THIS IS A PROPOSAL FOR EXTENDING "GUIENV", based on previous
   * discussions in the "GUIENV" design "team".
   *
   * The intended usage it for temporary drawings (i.e. non-permanent
   * in the sence of not automatic refresh etc., and non-interactive).
   * Can be used e.g. in the definition of borders, etc. on
   * interfaceObjects (if you make the graphics drawn on each refresh
   * of the interfaceObject).  Can be used for decorations, and for
   * grouping interfaceObject by enclosing them in a box etc.
   *)
  (# <<SLOT graphicsLib: attributes>>;

     overrideChildren:<
       (* If overrideChildren is furtherbound to return
        * TRUE, the drawings will overlay the children
        * of THIS(windowItem)
        *)
       booleanValue;
     pen: @
       (* defines the besic characteristics of the pen used for
        * drawing
        *)
       (# size:
            (# value: @integer;
            enter value do ...
            #);
          foregroundColor:
            (# theColor: @color;
            enter theColor do ...
            #);
          backgroundColor:
            (# theColor: @color;
            enter theColor do ...
            #);
          stipple:
            (# b: ^pixmap;
            enter b[] do ...
            #);
          mode:
            (* The transfer mode use specifies how new graphics are
             * mixed with the graphics already in the window
             *)
            (# m: @integer;
            enter m
            do ...
            #);
       #);
     style:
       (* The textstyle used for drawing text.*)
```

```
  (# theTextStyle: ^textStyle
  enter theTextStyle[]
  do ...;
  #);
move:
  (* move the pen to current position"+"p, without drawing
   * anything
   *)
  (# p: @point
  enter p do ...
  #);
moveTo:
  (* move the pen to position p, without drawing anything *)
  (# p: @point
  enter p do ...
  #);
draw:
  (* move the pen to current position"+"p, drawing a straigh
   * line between current position and the new position
   *)
  (# p: @point
  enter p do ...
  #);
drawTo:
  (* move the pen to position p, drawing a straigh line between
   * current position and the new position
   *)
  (# p: @point
  enter p do ...
  #);
drawSpot:
  (* Draws a single point *)
  (# p: @point
  enter p do ...
  #);
drawSpots:
  (* Draw an Array of points *)
  (# points: [0] ^point;
  enter points
  ...
  #);
drawLine:
  (* draws a line from p1 to p2. The pen position is not
   * affected
   *)
  (# p1, p2: @point
  enter (p1,p2) do ...
  #);
drawText:
  (* Draws the text from the current pen−position, using the
   * drawing characteristics of the pen (tile, color etc.)
   *)
  (# t: ^text
  enter t[] do ...
  #);
drawPolygon:
  (# points: [3] ^point
  enter points do ...
  #);
drawRect:
  (# r: @rectangle
  enter r do ...
  #);
drawRoundRect:
  (# r: @rectangle; roundness: @rectangle
  enter (r, roundness) do ...
```

```
      #);
   drawOval:
      (# r: @rectangle
      enter r do ...
      #);
   drawSlice:
      (# r: @rectangle; fromAngle, toAngle: @integer
      enter (r, fromAngle, toAngle) do ...
      #);
   fillPolygon:
      (# points: [3] ^point
      enter points do ...
      #);
   fillRect:
      (# r: @rectangle
      enter r do ...
      #);
   fillRoundRect:
      (# r: @rectangle; roundness: @rectangle
      enter (r, roundness) do ...
      #);
   fillOval:
      (# r: @rectangle
      enter r do ...
      #);
   fillSlice:
      (# r: @rectangle; fromAngle, toAngle: @integer
      enter (r, fromAngle, toAngle) do ...
      #);
   drawRaster:
      (# p: ^pixmap;
         from,to: @point;
         width,height: @integer;
      enter (p[],from,to,width,height) do ...
      #);
   private: @...;
do ...;
#)
```

20.7 Graphmath Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'guienvgraphmath' '../lib';
BODY 'private/graphmathbody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
-- lib: attributes --
point:
  (* A point is defined as the intersection between a vertical line
   * and a horisontal line in the coordinate plane
   *)
  (# <<SLOT pointLib: attributes>>;
     v, h: @integer;
     add:
       (* adds the coordinates of p to the coordinates THIS(point) *)
       (# p: @point
       enter p
       do ...
       #);
     subtract:
       (* subtracts the coordinates of p from the coordinates of
        * THIS(point)
        *)
       (# p: @point
       enter p
       do ...
       #);
     isEqual: booleanValue
       (* compares THIS(point) to p and returns true if they are
        * equal or false if not
        *)
       (# p: @point
       enter p
       do ...
       #);
  enter (h, v)
  exit  (h, v)
  #) (* point *);
rectangle:
  (* rectangles are used to define areas on the screen, to assign
   * coordinate systems to graphic entities, and to specify the
   * location and sizes for various drawing commands.  A rectangle is
   * defined by two points topLeft, bottomRight, which denote the
   * top-left corner and the bottom-right corner of the rectangle
   *)
  (# <<SLOT rectangleLib: attributes>>;
     topLeft:
       (#
       enter (left, top)
       exit (left, top)
       #);
     bottomRight:
       (#
       enter (right, bottom)
       exit (right, bottom)
       #);
     left, top, right, bottom: @integer;

     set:
       (* assigns the four boundary coordinates to THIS(rectangle) *)
```

```
  (# left, top, right, bottom: @integer
  enter (left, top, right, bottom)
  do ...
  #);
setFromPoints:
  (* sets THIS(rectangle) to the smallest rectangle that
   * encloses the two given points p1, p2
   *)
  (# p1, p2: @point;
  enter (p1, p2)
  do ...
  #);
size:
  (* evaluate the enter part to set the width and height.
   * Evaluate the exit part to get the width and height
   *)
  (# w, h: @integer;
  enter (#  enter (w, h) do ... #)
  exit (# do ... exit (w, h)  #)
  #);
offset:
  (* moves THIS(rectangle) by adding delta.h to each horizontal
   * coordinate and delta.v to each vertical coordinate
   *)
  (# delta: @point
  enter delta
  do ...
  #);
inset:
  (* shrinks or expands THIS(rectangle). The left and right
   * sides are moved in by the amount specified by delta.h; the
   * top and bottom are moved toward the center by the amount
   * specified by delta.v.  If delta.h or delta.v is negative,
   * the appropriate pair of sides is moved outward instead of
   * inward
   *)
  (# delta: @point
  enter delta
  do ...
  #);
intersection: booleanValue
  (* calculates the rectangle that is the intersection of src1
   * and src2, sets THIS(rectangle) to the intersection. Result
   * is set to true iff src1 and src2 indeed intersect
   *)
  (# src1, src2: @rectangle
  enter (src1, src2)
  do ...
  #);
union:
  (* calculates the smallest rectangle that encloses src1 and
   * src2, and sets THIS(rectangle) to the result
   *)
  (# src1, src2: @rectangle
  enter (src1, src2)
  do ...
  #);
containsPoint: booleanValue
  (* determines whether the pixel below and to the right of the
   * given coordinate point is enclosed in the specified
   * rectangle, and returns true if so or false if not
   *)
  (# p: @point
  enter p
  do ...
  #);
```

```
      pToAngle:
        (* calculates an integer angle between a line from the center
         * of the rectangle to thePoint and a line from the center of
         * the rectangle pointing straight up (12 o'clock high). The
         * angle is in degrees from 0 to 359, measured clockwise from
         * 12 oUclock, with 90 degrees at 3 o'clock, 180 at 6 o'clock,
         * and 270 at 9 o'clock
         *)
        (# thePoint: @point; angle: @integer
        enter thePoint
        do ...
        exit angle
        #);
      isEqual: booleanValue
        (* compares theRectangle to THIS(rectangle) and returns true
         * if they are equal or false if not. The two rectangles must
         * have identical boundary coordinates to be considered equal
         *)
        (# theRectangle: @rectangle
        enter theRectangle
        do ...
        #);
      isEmpty: booleanValue
        (* returns true if THIS(rectangle) is an empty rectangle or
         * false if not.  A rectangle is considered empty if the bottom
         * coordinate is less than or equal to the top or the right
         * coordinate is less than or equal to the left
         *)
        (# do ... #);
    enter (topLeft, bottomRight)
    exit  (topLeft, bottomRight)
    #) (* rectangle *);
  matrix:
    (# <<SLOT matrixLib: attributes>>;
      (* a  b  0
       * c  d  0
       * tx ty 1
       *)
      a, b, c, d, tx, ty: @real;
      inverse: ^matrix;
      mult: (* Multiply two matrices *)
        (# A, B, res: ^matrix;
        enter (A[], B[])
        do ...
        exit res[]
        #);
      transformPoint: @
        (# p, result: @point;
        enter p
        do ...
        exit result
        #);
      inverseTransformPoint: @
        (# p1, p2: @point;
        enter p1
        do ...
        exit p2
        #);
      transformRectangle: @
        (# r, result: @rectangle;
        enter r
        do ...
        exit result
        #);
      inverseTransformRectangle:
        (# r, result: @rectangle;
```

```
      enter r
      do ...
      exit result
      #);
    getInverse: @
      (# get: @...;
      do get;
      exit inverse[]
      #);
  enter (a, b, c, d, tx, ty)
  do INNER;
  exit (a, b, c, d, tx, ty)
  #);
IDmatrix:
  (# ID: ^matrix
  do ...
  exit ID[]
  #);
moveMatrix: matrix   (* A matrix specifying a translation *)
  (# itx, ity: @integer;
  enter (itx, ity)
  do ...
  #);
scaleMatrix: matrix  (* A matrix specifying a scaling *)
  (#
  enter (a, d)
  do ...
  #);
rotateMatrix: matrix (* A matrix specifying a rotation *)
  (# theta: @real;
  enter theta
  do ...
  #);
ovalAngle:
  (* Returns the angle a (in radians) and cos(a), sin(a), assuming
   * that (x,y) is a point on the oval with center in (cx,cy) and
   * horizontal radius hr and verticalradius vr, i.e.
   *    (x,y) = (cx,cy) + (hr*cos(a),vr*sin(a))
   *)
  (# cx, cy, hr, vr, x, y: @integer;
     a, cos_a, sin_a: @real;
     angle: @...;
  enter (cx, cy, hr, vr, x, y)
  do angle
  exit (a, cos_a, sin_a)
  #);
circleAngle:
  (* Returns the angle a (in radians) and cos(a), sin(a), assuming
   * that (x,y) is a point on the circle with center in (cx,cy) and
   * radius r, for some r i.e. (x,y) = (cx,cy) + (r*cos(a),r*sin(a))
   *)
  (# cx, cy, x, y: @integer;
     a, cos_a, sin_a: @real;
     angle: @...;
  enter (cx, cy, x, y)
  do angle
  exit (a, cos_a, sin_a)
  #);
region:
  (* A region is a collection of spatially coherent points *)
  (# <<SLOT regionLib: attributes>>;
     bounds:
       (# theRectangle: @rectangle;
       do ...
       exit theRectangle
       #);
```

```
allocate:
  (* allocates space for a new, variable−size region,
   * initializes it to the empty region defined by the rectangle
   * (0, 0)(0, 0)
   *)
  ...;
dispose:
  (* releases the memory occupied by THIS(region). Use this only
   * after you are completely through with a temporary region
   *)
  ...;
empty:
  (* destroys the previous structure of the given region, then
   * sets THIS(region) new to the empty region.
   *)
  ...;
setFromRectangle:
  (* destroys the previous structure of THIS(region), and then
   * sets the new structure to the rectangle specified by
   * theRectangle
   *)
  (# theRectangle: @rectangle
  enter theRectangle
  do ...
  #);
offset:
  (* moves THIS(region) on the coordinate plane, a distance of
   * delta.h horizontally and delta.v vertically
   *)
  (# delta: @point
  enter delta
  do ...
  #);
inset:
  (* shrinks or expands THIS(region). All points on the region
   * boundary are moved inwards a distance of dv vertically and
   * dh horizontally; if dh or dv is negative, the points are
   * moved outwards in that direction.  It leaves THIS(region)
   * centered at the same position, but moves the outline in −
   * for positive values of dh and dv − or out − for negative
   * values of dh and dv
   *)
  (# delta: @point
  enter delta
  do ...
  #);
intersection:
  (* calculates the intersection of two regions src1 and src2,
   * and sets THIS(region) to the intersection. This does not
   * create THIS(region); space must already have been allocated
   * for it.  THIS(region) can be one of the source regions, if
   * desired
   *)
  (# src1, src2: ^region
  enter (src1[], src2[])
  do ...
  #);
union:
  (* calculates the union of two regions src1 and src2, and sets
   * THIS(region) to the union. This does not create
   * THIS(region); space must already have been allocated for
   * THIS(region).  THIS(region) can be one of the source
   * regions, if desired
   *)
  (# src1, src2: ^region
  enter (src1[], src2[])
```

```
      do ...
      #);
    difference:
      (* subtracts src2 from src1 and sets THIS(region) to the
       * difference.  This does not create THIS(region); space must
       * already have been allocated for it. THIS(region) can be one
       * of the source regions, if desired
       *)
      (# src1, src2: ^region
      enter (src1[], src2[])
      do ...
      #);
    symDiff:
      (* calculates the difference between the union and the
       * intersection of src1 and src2 and places the result in
       * dstRgn.  This does not create THIS(region); space must
       * already have been allocated for it. THIS(region) can be one
       * of the source regions, if desired
       *)
      (# src1, src2: ^region
      enter (src1[], src2[])
      do ...
      #);
    containsPoint: booleanValue
      (* checks whether the pixel below and to the right of pt is
       * within THIS(region), and returns true if so or false if not
       *)
      (# pt: @point
      enter pt
      do ...
      #);
    containsRectangle: booleanValue
      (* checks whether theRectangle intersects the specified
       * region, and returns true if the intersection encloses at
       * least one bit or false if not
       *)
      (# theRectangle: @rectangle
      enter theRectangle
      do ...
      #);
    isEqual: booleanValue
      (* compares THIS(region) to theRegion and returns true if they
       * are equal or false if not. THIS(region) and theRegion must
       * have identical sizes, shapes, and locations to be considered
       * equal.  If THIS(region) and theRegion are empty regions true
       * is returned as well
       *)
      (# theRegion: ^region
      enter theRegion
      do ...
      #);
    isEmpty: booleanValue
      (* returns true if THIS(region) is an empty region or false if
       * not
       *)
      (# do ... #);
    private: @...;
  enter (# r: ^region  enter r[] do ... #)
  exit  (# r: ^region do ... exit r[] #)
  #)
```

20.8 Guienv Interface

```
ORIGIN '~beta/basiclib/betaenv';
INCLUDE '~beta/containers/list'
        'graphmath'
        'keys';
LIB_DEF 'guienv' '../lib';
BODY 'private/guienvbody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
-- lib: Attributes --
GUIenv:
  (#
     <<SLOT guienvLib:Attributes>>;
     onStartApplication:<
     (* is called when this application is started with no
      * documents. You can for example further bind this to show a
      * splash screen
      *) (# do INNER #);
     onOpenDocument:<
     (* is called whenever a user opens a document created by this
      * application
      *) (# fileName: ^text;  enter fileName[] do INNER #);
     onQuit:<
     (* is called when application is going to quit, either
      * because terminate is called or because the system are
      * are going to shut down.
      * If okToQuit is set to false the application will
      * not quit.
      *)
       (# okToQuit: @boolean;
       do true->okToQuit; INNER ;
       exit okToQuit
       #);
     onKeyDown:<
     (* is called when the user presses a key. This global keydown handler
      * is called before the normal event-dispatch mechanism is used. This
      * gives the application a chance to handle the key event in some special
      * way.
      *
      * Set done to true to stop the application from handling the key event in
      * the normal way.
      *)
       (# control: @boolean; key: @int32u; done: @boolean;
       enter (control,key)
       do INNER ;
       exit done
       #);
     terminate:
     (* will terminate the entire application if invoked
      * Terminate calls onQuit and will only quit if
      * onQuit returns true.
      *) (#  ... #);
     applicationMenubar:
     (* applicationMenubar is used to install a menubar with
      * functionality that is common for all parts for the
      * application.
      *)
       (# theMenubar: ^menubarType
       enter (#  enter theMenubar[] ... #)
       exit
```

```
    (#
    ...
    exit theMenuBar[]
    #)
  #);
menubarType:<
(* if further bound, an instance of menubarType is
 * automatically installed for the application. Further bind it
 * to standardMenubar if you want the standard menubar (file
 * and edit menu)
 *) menubar;
interfaceObject:
(* superpattern for all objects used for interaction with the
 * user
 *)
  (#
     <<SLOT interfaceObjectLib:Attributes>>;
     theEventhandler:
     (* The only instance of the eventhandler virtual *) @eventhandler;
     eventhandler:<
     (* Encapsulates the patterns related to event handling *)
       (#
          <<SLOT eventhandlerLib:Attributes>>;
          event:
          (* the abstract superpattern of all events *)
            (# <<SLOT eventLib:Attributes>>;
            ...
            #)
            (* event *)
            ;
          basicEvent: event
          (* abstract superpattern for all events
           * originating directly from the OS
           *)
            (#
               <<SLOT basicEventLib:Attributes>>;
               shiftKey: booleanValue
               (* true if the shiftkey was the down, when
                * THIS(basicEvent) occurred
                *) (#  ... #);
               altKey:
                booleanValue
               (* true if the altkey was the down, when
                * THIS(basicEvent) occurred
                *) (#  ... #);
               metaKey: booleanValue
               (* true if the metakey was the down, when
                * THIS(basicEvent) occurred
                *) (#  ... #);
               controlKey:
                booleanValue
               (* true if the controlkey was the down, when
                * THIS(basicEvent) occurred
                *) (#  ... #);
               buttonState:
                integerValue
               (* the number designating the button, which was
                * pressed down, when THIS(basicEvent) occurred
                * − 0 means 'no button'.  This value depe nds
                * on the number of buttons on the mouse −
                * Typically 1, 2 or 3.
                *) (#  ... #);
               when:
                integerValue
               (* the tick count when THIS(basicEvent)
                * occurred.  1 tick = 1/60 sec.
```

```
      *) (#  ... #);
     globalPosition:
     (* global coordinates of the mouse, when
      * THIS(basicEvent) occurred
      *)
       (# p: @point;
       ...
       exit p
       #);
     localPosition:
     (* local coordinates of the mouse, when
      * THIS(basicEvent) occurred − relative to
      * THIS(inteefaceObject)
      *)
       (# p: @point;
       ...
       exit p
       #);
  do INNER ;
  #);
mouseEvent:
 basicEvent
(* abstract superpattern for events related to the
 * mouse
 *)
  (#
     <<SLOT mouseEventLib:Attributes>>;
     doubleClick: booleanValue
     (* true if THIS(mouseEvent) is a doubleclick.
      * For a mouse click to qualify as doubleclick
      * it must happen close in time and space, and
      * with the same mouse button
      *) (#  ... #);
  do INNER ;
  #);
keyEvent: basicEvent
(* abstract superpattern for events related to the
 * keyboard.
 *)
  (#
     <<SLOT keyEventLib:Attributes>>;
     ch:
     (* the key on the keyboard, related to
      * THIS(keyEvent)
      *)
       (# theChar: @char;
       ...
       exit theChar
       #);
     key:
     (* the specialkey on the keyboard, related to
      * THIS(keyEvent).
      * See keys.bet for descriptions.
      *)
       (# theKey: @int32;
       ...
       exit theKey
       #)
  do INNER
  #);
mouseDown: mouseEvent
(* This event occurs when the user presses any mouse
 * button down on THIS(interfaceObject)
 *)
  (#
     <<SLOT mouseDownLib:Attributes>>;
```

```
            delay:
            (* used to wait for period ticks to pass, while
             * mouse.isStillDown is true, and then execute
             * INNER.  If mouseStillDown becomes false
             * before period ticks, INNER is not executed
             *)
              (# period: @integer
              enter period
              ...
              #)
        do INNER
        #);
      onMouseDown:< mouseDown;
      mouseUp: mouseEvent
      (* This event occurs when the user releases any
       * mouse button after having pressed it on
       * THIS(interfaceObject)
       *) (# do INNER #);
      onMouseUp:< mouseUp;
      keyDown: keyEvent
      (* Occurs when the user presses a key, related to
       * THIS(interfaceObject)
       *) (# do INNER #);
      onKeyDown:< keyDown;
      refresh: basicEvent
      (* This event tells THIS(interfaceobject), that it
       * needs to redraw itself. UpdateRect is the
       * rectangle that needs to be updated expressed
       * in the coordinate system of this(interfaceObject).
       *)
        (#
           updateRect:
             (# value: ^rectangle;
             ...
             exit value[]
             #)
        do INNER
        #);
      onRefresh:< refresh;
      activate: basicEvent
      (* Send when THIS(interfaceObject) becomes active *)
        (# do INNER #);
      onActivate:< activate;
      deactivate: basicEvent
      (* Send when THIS(interfaceObject) becomes inactive
       *) (# do INNER #);
      onDeactivate:< deactivate;

  #);
action:
(* Actions is a means of subscribing to events.  The
 * desired event is specified by further binding
 * eventType.  Actions can be prepended or appended to
 * THIS(interfaceobject).  When some event is called, the
 * prepended actions for the event is called *before* the
 * INNER and the appended actions are called after.
 *)
  (#
     <<SLOT actionLib:Attributes>>;
     eventType:< theEventhandler.event;
     theEvent: ^eventType;

  enter theEvent[]
  do INNER ;
  #);
prependAction:
```

```
(* Prepends the action, so it will be executed before the
 * event is subscribes to
 *)
  (# theAction: ^action;
  enter theAction[]
  ...
  #);
appendAction:
(* Appends the action, so it will be executed after the
 * event is subscribes to.
 *)
  (# theAction: ^action;
  enter theAction[]
  ...
  #);
deleteAction:
(* Remove the action *)
  (# theAction: ^action;
  enter theAction[]
  ...
  #);
open:<
(* must be called before any other operation on
 * THIS(interfaceObject).
 *)
  (#
     create:< (#  ... #);
  ...
  #);
close:<
(* closes THIS(interfaceObject) and dispose all related
 *              structures
 *) (#  ... #);
enableEventType:<
(* makes THIS(interfaceObject) sensible to the specified
 * type of events
 *)
  (# ev: ##theEventhandler.event
  enter ev##
  ...
  #);
disableEventType:<
(* makes THIS(interfaceObject) insensible to the
 * specified type of events
 *)
  (# ev: ##theEventhandler.event
  enter ev##
  ...
  #);
interfaceObjectException:
 exception
(* abstract superpattern for exceptiosn related to
 * THIS(interfaceObject).
 *) (#  ... #);
notOpenedException:
 interfaceObjectException
  (# location: ^text
  enter location[]
  ...
  #);
notOpenedError:<
(* this exception is raised if any operation is performed
 * on THIS(interfaceObject) is called before open is
 * called.  This will also happen if "close" is called
 * twice
 *) notOpenedException;
```

```
      private: @...;

    do INNER
    #)
    (* interfaceObject *)
    ;
menubar: interfaceObject
(* menubar is a bar containing the titles of the contained
 * menus. A menu is pulled down by clicking at the title,
 * allowing the user to select a menuitem in the menu.  A
 * menubar is only visible if it is installed − either as the
 * global menubar or as the menubar in some window.
 *)
  (#
     <<SLOT menubarLib:Attributes>>;
     append:
     (* inserts a menu after all menues in the menubar.  If
      * the menu is already in the menu bar, nothing happens
      *)
       (# theMenu: ^menu;
       enter theMenu[]
       ...
       #);
     delete:
     (* deletes a menu from the menu bar. The menu titles
      * following the deleted menu will move over to fill the
      * vacancy
      *)
       (# theMenu: ^menu
       enter theMenu[]
       ...
       #);
     clear:
     (* removes all menues from the menu bar when you want to
      * start with new menues
      *) (#  ... #);
     appendMenubar:
     (* inserts all menues in another menubar after all menues
      * in THIS(menubar).  This is the same as calling
      * insertMenubar with NONE as afterMenu.
      *)
       (# theMenubar: ^menubartype;
       enter theMenubar[]
       ...
       #);
     replaceMenubar:
     (* replace all menues in theMenubar with all menues in
      * replacementMenubar in THIS(menubar).
      *)
       (# theMenubar,replacementMenubar: ^menubartype
       enter (theMenubar[],replacementMenubar[])
       ...
       #);
     deleteMenubar:
     (* deletes all menues in theMenubar from
      * THIS(menubar). The menu titles following the menues in
      * the deleted menubar will move over to fill the vacancy
      *)
       (# theMenubar: ^menubartype
       enter theMenubar[]
       ...
       #);
     scan:
     (* iterates over all menues currently inserted in the
      * menubar
      *) (# current: ^menu;  ... #);
```

```
      open::<
        (#
           create::<
             (#
             ...
             #);

        ...
        #);
      close::<
        (#
        ...
        #);
      private: @...;

   #)
   (* menubar *)
   ;
menu: interfaceObject
(* menu contains a group of menuitems and is usefull for
 * letting the user perform commands or set settings in the
 * application. A menu can be installed in a menubar, as a
 * submenu to some menuitem or simply be popped up on the
 * screen.
 *)
   (#
      <<SLOT menuLib:Attributes>>;
      name:
      (* the name of the menu as shown in the menubar.  if the
       * menu is not in a menubar, the name is not visible
       *)
        (# theName: ^text
        enter (#  enter theName[] ... #)
        exit
          (#
          ...
          exit theName[]
          #)
        #);
      eventhandler::<
        (#
           select: event
           (* executed when the user selects THIS(menu) (or
            * pops it up) just before the menu is shown.
            *) (# do INNER #);
           onSelect:< select;

        #);
      menuitem: interfaceObject
      (* menuitem is used for letting the user perform commands
       * in the application or display the state of some option,
       * by checking and unchecking the menuitem. It can also
       * serve as the title of a submenu.
       *)
        (#
           <<SLOT menuitemLib:Attributes>>;
           key:
           (* the key shortcut of THIS(menuitem), allows the
            * user to select THIS(menuitem) without using the
            * mouse.
            *)
             (# c: @char
             enter (#  enter c ... #)
             exit
               (#
               ...
```

```
    exit c
    #)
  #);
specialkey:
(* Extendend version of key.
 * key is the shortcut, if less than 255 used as char.
 *)
  (#
     key: @integer;
     (* from special keys in keys.bet *)
     ctrl,shift,alt: @boolean;

  enter
    (#
    enter (key,shift,ctrl,alt)
    ...
    #)
  #);
name:
(* models the name of THIS(menuitem).  Evaluate the
 * enter−part to set the name.  Evaluate the
 * exit−part to get the name
 *)
  (# t: ^text;
  enter
    (#  enter t[] ... #)
  exit
    (#
    ...
    exit t[]
    #)
  #);
checked:
(* when THIS(menuitem) is checked, a check mark is
 * displayed at the left side the menuitem
 *)
  (# checked: @boolean
  enter
    (#
    enter checked
    ...
    #)
  exit
    (#
    ...
    exit checked
    #)
  #);
subMenu:
(* if a submenu is attached to THIS(menuitem), that
 * menu is pulled down by selecting
 * THIS(menuitem). In that case onSelect is never
 * issued for THIS(menuitem)
 *)
  (# theMenu: ^menu;
  enter
    (#  enter theMenu[] ... #)
  exit
    (#
    ...
    exit theMenu[]
    #)
  #);
position: IntegerValue
(* the position of THIS(menuitem) in its menu,
 * separator items are counted as well
```

```
      *) (#  ... #);
     eventhandler::<
       (#
          onStatus:<
           booleanValue
          (* executed just before THIS(menuitem) is
           * shown.  should return true if THIS(menuitem)
           * is enabled.  Default is true
           *) (#  ... #);
          select: event
          (* executed when THIS(menuitem) is selected in
           * the menu.  If a submenu is attached, it will
           * not be executed − instead the submenu is
           * pulled down
           *) (# do INNER #);
          onSelect:< select;

       #);
     open::<
       (#
          create::<  (#  ... #);
       ...
       #);
     private:
       @...;

  do INNER
  #)
  (* menuitem *)
  ;
dynamicMenuitem: menuitem
(* dynamic menuitem does not call its own onStatus and
 * onSelect events, instead these events are called on the
 * attached action, if any is attached
 *)
  (#
     <<SLOT dynamicItemLib:Attributes>>;
     theAction: ^menuAction;
     attach:
     (* anAction is attached to THIS(menuitem) *)
       (# anAction: ^menuAction;
       enter anAction[]
       ...
       #);
     detach:
     (* the menuitemHandler that is currently attached to
      * THIS(menuitem) is detached, meaning that no action
      * is attached
      *) (#  ... #);
     eventhandler::<
       (#
          onStatus::<
            (#
            ...
            #);
          onSelect::<
            (#  ... #);

       #);
  #)
  (* dynamicMenuitem *)
  ;
menuAction:
(* a menuAction can dynamicly be attached to
 * dynamicMenuitems within THIS(menu), meaning that the
 * onStatus and onSelect events of THIS(menuAction) will
```

```
 * be executed instead of these events of the
 * dynamicMenuitem. The pointer "theMenuitem" refers to
 * the dynamicMenuitem THIS(menuAction) is currently
 * attached to
 *)
  (#
     theMenuitem: (* the menuitem THIS(menuAction) is attached to *)
       ^dynamicMenuitem;
     onStatus:< booleanValue
     (* this status is evaluated instead of the status of
      * the actual menuitem (theMenuitem) THIS(menuAction)
      * is attached to.  Default returns true
      *) (#  ... #);
     onSelect:<
     (* onSelect is executed from the hit of the actual
      * dynamicMenuitem THIS(action) is attached to
      *) object;

  #)
  (* action *)
  ;
separator: menuitem
(* defines a menu separator, which is a unselectable line
 * in the menu, dividing groups of menuitems.
 *)
  (#
     open::<
       (#
          create::<  (#  ... #);

       ...
       #);
     close::<
       (#
       ...
       #);

  #);
append: (* appends the menuitem to THIS(menu) *)
  (# theMenuitem: ^menuitem
  enter theMenuitem[]
  ...
  #);
delete:
(* deletes the menuitem from THIS(menu) *)
  (# theMenuitem: ^menuitem
  enter theMenuitem[]
  ...
  #);
scan:
(* iterates over all menuitems in THIS(menu) *)
  (# current: ^menuitem ... #);
clear:
(* deletes all menuitems in  THIS(menu) *)
  (#  ... #);
noOfMenuitems: integerValue
(* returns the number of menuitems in THIS(menu) *)
  (#  ... #);
popUp:
(* THIS(menu) is popped up as follows: The menuitem
 * indexed by "popupWith" is selected (not checked but
 * hilited) and popupAt is the top left corner of that
 * menuitem in the coordinate system of the popupIn
 * window.
 *)
  (#
```

```
            popupWith: @integer;
            popupAt: @point;
            popupIn: ^window.windowitem;

          enter (popupWith,popupAt,popupIn[])
          ...
          #);
        getMenuitemByNumber:
        (* returns a reference to the menuitem at the specified
         * position in the menu
         *)
          (# number: @integer; theMenuitem: ^menuitem;
          enter number
          ...
          exit theMenuitem[]
          #);
        enable:
        (* enable THIS(menu) *) (#  ... #);
        disable: (* disable THIS(menu) *)
          (#  ... #);
        enabled:< booleanValue
        (* should return true if THIS(menu) is enabled *)
          (#  ... #);
        open::<
        (* the menu is not automatically inserted in the
         * menubar. You have to do this yourself
         *)
          (# create::<  (#  ... #);
          ...
          #);
        close::<
          (#  ... #);
        private:
          @...;

    #)
    (* menu *)
    ;
  standardMenubar: menubar (* idx+ *)
    (#
        standardFileMenu: menu
          (#
            newMenuitem: @dynamicMenuitem;
            openMenuitem: @dynamicMenuitem;
            closeMenuitem: @dynamicMenuitem;
            saveMenuitem: @dynamicMenuitem;
            saveAsMenuitem: @dynamicMenuitem;
            revertMenuitem: @dynamicMenuitem;
            printMenuitem: @dynamicMenuitem;
            pageSetUpMenuitem: @dynamicMenuitem;
            quitMenuitem: @dynamicMenuitem;
            open::<  (#  ... #);

          #)
          (* standardFileMenu *)
          ;
        fileMenu:< menu;
        theFileMenu: ^fileMenu;
        standardEditMenu: menu
          (#
            undoMenuitem:
              @dynamicMenuitem;
            cutMenuitem: @dynamicMenuitem;
            copyMenuitem: @dynamicMenuitem;
            pasteMenuitem: @dynamicMenuitem;
            clearMenuitem: @dynamicMenuitem;
```

```
            open::<  (#  ... #)
          #)
          (* standardEditMenu *)
          ;
        editMenu:< menu;
        theEditMenu: ^editMenu;
        open::<
          (#
          ...
          #);

    #);
  window: interfaceObject
  (* user interaction with the window such as dragging and
   * resizing is taken care of by the window manager.  Anything
   * visible you may want to place in the window is subpatterns
   * of the abstract pattern windowitem, which is a subpattern of
   * interfaceObject. The window can be used as a modal dialog by
   * means of the pattern "showModal"
   *)
    (#
       <<SLOT windowLib:Attributes>>;
       AutomaticTarget:< BooleanValue;
       eventhandler::<
         (#
            aboutToClose: event
            (* is called whenever the user has performed an
             * action that causes THIS(window) to close.  Further
             * bind this to perform actions before the window is
             * actually closed.  You can prevent the window from
             * closing by assigning false to the boolean
             * 'okToClose'
             *)
              (# okToClose: @boolean
              do true->okToClose; INNER
              exit okToClose
              #);
            onAboutToClose:< aboutToClose;
            onActivate::<
            (* is send to contents, which takes care of sending
             * the event to all children
             *)  (#  ... #);
            onDeactivate::<
            (* is send to contents, which takes care of sending
             * the event to all children
             *)  (#  ... #);

         #);
       theMenubar:
       (* is used to install a menubar for THIS(window), and to
        * gain access to the menubar of THIS(window)
        *)
         (# theBar: ^menubartype
         enter
           (#  enter theBar[] ... #)
         exit
           (#  ... exit theBar[] #)
         #);
       menubarType:<
       (* if further bound, an instance of menubarType is
        * automatically installed for THIS(window)
        *) menubar;
       menubarVisible:< (* Specifies if the menubar should be visible. *)
        trueObject;
       type:<
       (* The type can be one of the following:
```

```
 *   windowTypes.normal  <− default
 *   windowTypes.dialog
 *   windowTypes.palette
 *) integerValue;
resizeable:< booleanValue
  (#
  do (if type = windowTypes.normal then true−>value; INNER if);
  #);
title:
(* the title of the window is displayed in the windows
 * title−bar if the window has one.
 *)
  (# theTitle: ^text
  enter
    (#  enter theTitle[] ... #)
  exit
    (#
    ...
    exit theTitle[]
    #)
  #);
position:
(* the window's position is the coordinates of the
 * topLeft corner of the window's inside rectangle on the
 * screen
 *)
  (# pt: @point;
  enter (#  enter pt ... #)
  exit
    (#
    ...
    exit pt
    #)
  #);
size:
(* the size is the size of the inside rectangle of the
 * window
 *)
  (# width,height: @integer;
  enter
    (#  enter (width,height) ... #)
  exit
    (#  ... exit (width,height) #)
  #);
frame:
(* the frame is defined as the rectangle THIS(window)
 * occupies on the screen = (position,position + size)
 *)
  (# theFrame: @rectangle;
  enter (#  enter theFrame ... #)
  exit (#  ... exit theFrame #)
  #);
insideRectangle:
(* the inside rectangle is the window's content rectangle
 * in terms of local coordinates in the window.  The top
 * left corner is (0, 0) and the bottom right corner is
 * the window's size
 *)
  (# theRectangle: @rectangle;
  ...
  exit theRectangle
  #);
show:
(* shows THIS(window) in front of other windows *)
  (#  ... #);
showModal:
```

```
(* shows THIS(window) in a modal way. Interaction with
 * other windows is prevented until THIS(window) is either
 * closed or hidden, and then showModal returns to the
 * caller
 *) (#  ... #);
hide:
(* hides THIS(window), i.e. make it invisible without
 * destroying it. Can be made visible again using show
 *) (#  ... #);
visible:
(* The visibility of the window. *)
  (# value: @boolean;
  enter (#  enter value ... #)
  exit
    (#
    ...
    exit value
    #)
  #);
maxSize:
(* use this to set the maximum size THIS(window) is
 * allowed to get, when resized by the user. maxSize
 * doesn't affect the behaviour of setSize.
 *)
  (# width,height: @integer;
  enter
    (#
    enter (width,height)
    ...
    #)
  exit
    (#
    ...
    exit (width,height)
    #)
  #);
minSize:
(* use this to set the minimum size THIS(window) is
 * allowed to get, when resized by the user. minSize
 * doesn't affect the behaviour of setSize
 *)
  (# width,height: @integer;
  enter
    (#
    enter (width,height)
    ...
    #)
  exit
    (#
    ...
    exit (width,height)
    #)
  #);
bringToFront:
(* THIS(window) is brought to the front of all other
 * windows
 *) (#  ... #);
bringBack:
(* THIS(window) is placed behind all other windows *)
  (#  ... #);
bringBehind:
(* THIS(window) is placed behind the window referred to
 * by "theWindow"
 *)
  (# theWindow: ^window;
  enter theWindow[]
```

```
  ...
  #);
update:
(* Updates the window by posting a refresh event.  If
 * emmediate is true, the refresh event will be processed
 * immediately.
 *)
  (# immediate: @boolean;
  enter immediate
  ...
  #);
backgroundColor:
(* Sets backgroundcolor of this window *)
  (# theColor: @color
  enter theColor
  ...
  #);
contents:
(* The contents of THIS(window) is the father of all
 * other windowitems in THIS(window).
 *)
  (# theContents: ^canvas;
  ...
  exit theContents[]
  #);
target:
(* the window's target is a reference to the windowitem
 * that receives keyDown. You are responsible for making
 * sure the window's target is the windowitem that is
 * affected by menu commands. The eventhandler of
 * windowitem has two events: "enableTarget" and
 * "disableTarget". When a windowitem is becomming the new
 * target, first "disableTarget" is called for the old
 * target then "enableTarget" is called for the new target
 *)
  (# theTarget: ^windowitem;
  enter
    (#  enter theTarget[] ... #)
  exit
    (#
    ...
    exit theTarget[]
    #)
  #);
windowitem: interfaceObject
(* superclass for all interfaceobjects in this window.  A
 * windowitem is always part of a canvas (father)
 *)
  (#
     <<SLOT windowitemLib:Attributes>>;
     automaticTarget:< BooleanValue;
     eventhandler::<
       (#
          visibleChanged: event
          (* is called, when THIS(windowitem) is hidden
           * or shown
           *) (# do INNER #);
          onVisibleChanged:< visibleChanged;
          frameChanged: event
          (* is called whenever the frame of
           * THIS(windowitem) is changed
           *)
            (# oldFrame,newFrame: @rectangle;
            enter (oldFrame,newFrame)
            do INNER
            #);
```

```
      onFrameChanged:< frameChanged;
      fatherFrameChanged: event
      (* is called when the frame of the father of
       * THIS(windowitem) is changed
       *)
        (# oldFrame,newFrame: @rectangle;
        enter (oldFrame,newFrame)
        do INNER
        #);
      onFatherFrameChanged:< fatherFrameChanged;
      enabledChanged: event
      (* is called, when THIS(windowitem) is
       * enabled/disabled
       *) (# do INNER #);
      onEnabledChanged:< enabledChanged;
      enableTarget: event
      (* is called when THIS(windowitem) is becomming
       * target in the window
       *) (# do INNER #);
      onEnableTarget:< enableTarget;
      disableTarget: event
      (* is called when THIS(windowitem) was target
       * and another windowitem is becomming target
       *) (# do INNER #);
      onDisableTarget:< disableTarget;
      borderVisibleChanged: event
      (* is called, when the border of
       * THIS(windowitem) is shown or hidden
       *) (# do INNER #);
      onBorderVisibleChanged:< borderVisibleChanged;
      borderStyleChanged: event
      (* is called, when the border style of
       * THIS(windowitem) is changed
       *) (# do INNER #);
      onBorderStyleChanged:< borderStyleChanged;
      theCursorChanged: event
      (* is called, when THIS(windowitem) is assigned
       * a new cursor
       *) (# do INNER #);
      onTheCursorChanged:< theCursorChanged;
      hiliteChanged: event
      (* Is called when THIS(windowitem) is hilited
       * or dehilited
       *) (#  do INNER ;  #);
      onHiliteChanged:< hiliteChanged;
      onRefresh::<
        (#  ... #);
      mouseEnter: event
      (* Is called when the mouse enters THIS(windowitem) *)
        (# do INNER #);
      onMouseEnter:< mouseEnter;
      mouseLeave: event
      (* Is called when the mouse leaves THIS(windowitem) *)
        (# do INNER #);
      onMouseLeave:< mouseLeave;

  #);
father: ^
(* father is the canvas that THIS(windowitem) is a
 * child of
 *) canvas;
frame:
(* the frame is defined as the rectangle
 * THIS(windowitem) occupies in the coordinate system
 * of the father. When the frame is changed
 * THIS(windowitem) is updated and the father is
```

```
 * informed about the change.  If you need other
 * actions to take place, when changing the frame,
 * you must further bind the event onFrameChanged
 *)
  (# theFrame: @rectangle;
  enter
    (#
    enter theFrame
    ...
    #)
  exit
    (#
    ...
    exit theFrame
    #)
  #);
position:
(* the position of THIS(windowitem) is defined as
 * the topLeft corner of the bounding frame. When the
 * position is changed, the frame is changed, so the
 * onFrameChanged event is called
 *)
  (# pt: @point;
  enter
    (#
    enter pt
    ...
    #)
  exit
    (#
    ...
    exit pt
    #)
  #);
move:
(* moves THIS(windowitem) relative (dh, dv), by
 * setting the position, meaning that the
 * onFrameChanged event is called
 *)
  (# dh,dv: @integer;
  enter (dh,dv)
  ...
  #);
size:
(* the size of THIS(windowitem) is defined as the
 * height and width of the bounding frame. When the
 * size is changed, the frame is changed, so the
 * onFrameChanged event is called
 *)
  (# width,height: @integer;
  enter
    (#
    enter (width,height)
    ...
    #)
  exit
    (#
    ...
    exit (width,height)
    #)
  #);
fitToContents:<
(* Adjusts the size of THIS(windowItem) to
 * fit the contents
 *)
  (# doneInInner: @boolean;
```

```
  ...
  #);
bindLeft,bindRight,bindBottom,
  bindTop: @
(* these attributes specify how THIS(windowitem)
 * shall behave when the father changes it's
 * frame. If e.g. "bindLeft" is true, the leftSide
 * will have the same constant distance to the
 * leftSide of the father, when the father is resized
 *) boolean;
visible:
(* an invisible windowitem will be ingored w.r.t.
 * user interaction (it is not visible on the screen)
 *)
  (# value: @boolean;
  enter
    (#
    enter value
    ...
    #)
  exit
    (#
    ...
    exit value
    #)
  #);
hilite:
  (# value: @boolean;
  enter
    (#
    enter value
    ...
    #)
  exit
    (#
    ...
    exit value
    #)
  #);
show: (* makes THIS(windowitem) visible *)
  (#  ... #);
hide:
(* makes THIS(windowitem) invisible *)
  (#  ... #);
enabled:
(* if THIS(windowitem) is enabled it receives mouse
 * events or key events
 *)
  (# value: @boolean
  enter
    (#
    enter value
    ...
    #)
  exit
    (#
    ...
    exit value
    #)
  #);
enable:
(* enables THIS(windowitem) so it can receive mouse
 * or key events
 *) (#  ... #);
disable:
(* disables THIS(windowitem) so it does not receive
```

```
 * any mouse or key events
 *) (#  ... #);
backgroundColor:
  (# theColor: @color;
  enter
    (#
    enter theColor
    ...
    #)
  exit
    (#
    ...
    exit theColor
    #)
  #);
border: @
(* the border around THIS(windowitem) makes it
 * apparent, where it is located on the screen.
 *) (* idx+ *)
  (#
     visible:
     (* if the border is visible, the insideRect of
      * THIS(windowitem) is inset depending on the
      * style of the border.
      *)
       (# value: @boolean;
       enter
         (#
         enter value
         ...
         #)
       exit
         (#
         ...
         exit
         value
         #)
       #);
     style:
     (* the border style can be one of the
      * following:
      *   borderStyles.simple:
      *      A simple one pixel wide border.
      *   borderStyles.shadowIn:
      *      Draws the border so THIS(windowitem)
      *      appears inset.
      *   borderStyles.shadowOut:
      *      Draws the border so THIS(windowitem)
      *      appears outset.
      *   borderStyles.etchedIn:
      *      Draws the border using a double line
      *      giving the effect of a line etched
      *      into the window.
      *   borderStyles.etchedOut:
      *      Draws the border using a double line
      *      giving the effect of a line comming
      *      out of the window.
      *)
       (# value: @integer;
       enter
         (#
         enter value
         ...
         #)
       exit
         (#
```

```
          ...
          exit
          value
          #)
        #);

  #);
insideRectangle:
(* insideRectangle is the area inside the border of
 * THIS(windowitem).
 *)
  (# theRectangle: @rectangle;
  ...
  exit
  theRectangle
  #);
theCursor:
(* theCursor is used to install a cursor for
 * THIS(windowitem), and to gain access to the cursor
 * of THIS(windowitem)
 *)
  (# theCur: ^cursor;
  enter
    (#
    enter theCur[]
    ...
    #)
  exit
    (#
    ...
    exit theCur[]
    #)
  #);
cursorType:<
(* if further bound, an instance of cursorType is
 * automatically installed for THIS(windowitem)
 *) cursor;
trackMouse:
(* this is a control pattern usually evaluated from
 * a mouseDown eventhandler. Initially 'mousePress'
 * is evaluated, then 'mouseMove' is evaluated
 * whenever the mouse moves as long as the mouse is
 * stillDown − (h, v) will be the horizontal and
 * vertical distance the mouse has moved since the
 * last call to 'mouseMove'.  When the user releases
 * the mouse, 'mouseRelease' is evaluated.  If the
 * mouse isn't stillDown (see stillDown) when track
 * is called nothing will happen.  All the
 * coordinates are local to THIS(WindowItem).
 *)
  (#
     mousePress:< object;
     mouseMove:<
       (# h,v: @integer;  enter (h,v) do INNER #);
     mouseRelease:< object;
     curPt,prevPt: @point;

  ...
  #);
drag:
(* lets the user drag a gray outline of this
 * windowitem
 *) (#  ... #);
resize:
(* lets the user resize this windowitem by dragging
 * a gray outline
```

```
       *) (#  ... #);
      update:
      (* THIS(windowitem) is updated, by posting an
       * refresh event to the window. If "immediate" is
       * true the update is performed immediately,
       * otherwise the update is performed, when there is
       * no other event waiting (this is normally what you
       * want)
       *)
        (# immediate: @boolean;
        enter immediate
        ...
        #);
      open::<
      (* initially a windowitem is visible and active *)
        (#
           create::<  (#  ... #);

        enter father[]
        ...
        #);
      close::<
      (* no actions are performed at this level *)
        (#  ... #);
      private:
        @...;

  #);
(* windowitem *)
separator: windowitem
(* a separator is a horzontal or vertical separating line
 *)
  (#
     <<SLOT separatorLib:Attributes>>;
     eventhandler::<
       (#
          styleChanged: event
          (* Called when the style is changed *)
            (# do INNER #);
          onStyleChanged:< styleChanged;
          onRefresh::<
            (#  ... #);

       #);
     vertical:<
     (* Further bind to specify the orientation of
      * THIS(separator) default is horizontal
      *) booleanObject;
     style:
     (* the  style can be one of the following:
      *   lineStyles.singleLine:
      *     A single line is drawn.
      *   lineStyles.doubleLine:
      *     A double line is drawn.
      *   lineStyles.dashedSingleLine:
      *     A dashed single line is drawn.
      *   lineStyles.dashedDoubleLine:
      *     A dashed double line is drawn.
      *   lineStyles.etchedIn:
      *     A double line is drawn giving the effect of
      *     a line etched into the window.
      *   lineStyles.etchedOut:
      *     A double line is drawn giving the effect of
      *     a line comming of of the window.
      *)
       (# value: @integer;
```

```
        enter
          (#  enter value ... #)
        exit
          (#  ... exit value #)
        #);
      open::<
        (#
           create::<
             (#  ... #);

        ...
        #);
      close::<  (#  ... #);
      private:
        @...;

  #);
canvas: windowitem
(* A canvas is a sub−window in the window.  Only the
 * windowitems located inside the frame of THIS(canvas)
 * will be visible
 *)
  (#
     <<SLOT canvasLib:Attributes>>;
     eventhandler::<
       (#
          childFrameChanged: event
          (* is called when a child of THIS(canvas) has
           * changed frame
           *)
            (# oldFrame,newFrame: @rectangle;
            enter (oldFrame,newFrame)
            do INNER
            #);
          onChildFrameChanged:< childFrameChanged;
          onActivate::<
            (#  ... #);
          onDeactivate::<
            (#
            ...
            #);
          onMouseDown::<
            (#  ... #);
          onRefresh::<
            (#
            ...
            #);
          onMouseUp::<
            (#  ... #);
          onFrameChanged::<
            (#
            ...
            #);
          onVisibleChanged::<
            (#  ... #);

       #);
     selection: @
       (#
          add:
            (# theWindowitem: ^windowitem;
            enter theWindowitem[]
            ...
            #);
          set:
            (# theWindowitem: ^windowitem;
```

```
            enter theWindowitem[]
            ...
            #);
          remove:
            (# theWindowitem: ^windowitem;
            enter theWindowitem[]
            ...
            #);
          empty: booleanValue
            (#  ... #);
          scan:
            (#
               current:
                 ^windowitem;

            ...
            #);
          clear:
            (#
            ...
            #);

      #);
    scan: (* Scan operation on the children of THIS(canvas) *)
      (# current: ^windowitem;
      ...
      #);
    open::<
    (* The canvas is opened and displayed. *)
      (# create::<  (#  ... #);
      ...
      #);
    close::<
    (* close is called for all the children of
     * THIS(canvas)
     *)  (#  ... #);
    private:
      @...;

  #)
  (* canvas *)
  ;
localToGlobal:
(* Translate the point from global coordinates to window
 * coordinates.
 *)
  (# local,global: @point;
  enter local
  ...
  exit global
  #);
globalToLocal:
(* Translates the point to window coordinates to global
 * local coordinates
 *)
  (# global,local: @point;
  enter global
  ...
  exit local
  #);
addTarget:
  (# item: ^windowitem
  enter item[]
  ...
  #);
open::<
```

```
      (# create::<  (#  ... #);
      ...
      #);
    close::<
    (* the windows close operation is normally automatically
     * called from the content's aboutToGoAway event.  You can
     * also call it directly. theContents.close is called to
     * close all of the windows internal structures
     *)  (#  ... #);
    <<SLOT BifrostAttributes:Attributes>>;
    private: @...;

  #)
  (* window *)
  ;
cursor:
(* A cursor is the raster attached to the mouse pointer *)
  (#
     <<SLOT cursorLib:Attributes>>;
     private:
       @...
  #);
pixmap: (* Pixmap pattern *)
  (#
     <<SLOT pixmapLib:Attributes>>;
     read:
     (* Reads the specified file into THIS(pixmap).
      * The type of the  file are guessed by looking
      * at the extension, or the the first few bytes,
      * or the macintosh file type − all depending
      * on the platform
      *)
       (#
          name: ^text;
          error:< exception
            (# what: ^text;
            enter what[]
            do what[]−>msg.append; INNER ;
            #);

       enter name[]
       ...
       #);
     clear:
     (* Clear the Pixmap with the specified color *)
       (# theColor: @Color;
       enter theColor
       ...
       #);
     writeJPG:
       (#
          name: ^Text;
          area: @rectangle;
          quality: @integer;
          (* 50 − 100 % *)

       enter (name[],area,quality)
       ...
       #);
     writePNG:
       (#
          name: ^Text;
          area: @rectangle;
          depth: @integer;
          (* 8, 24, or 32 *)
```

```
  enter (name[],area,depth)
  ...
  #);
init:<
(* Intializes the raster to have the specified width
 * and height. Allocates any data needed −
 * you have to call dispose to free that data.
 *)
  (# width,height: @integer;
  enter (width,height)
  ...
  #);
dispose:<
(* call this to dispose the memory occupied
 * by THIS(pixmap) when completely done with
 * THIS(pixmap)
 *) (#  ... #);
width: integerValue
(* returns the width set by init or by
 * read operations
 *) (#  ... #);
height: integerValue
(* returns the height set by init or
 * by read operations
 *) (#  ... #);
transparent:
(* Specifies that the "background" of this(pixmap) is
 * transparent.
 * This attribute is automatically set to TRUE
 * If a transparentColor or a mask is specified.
 * If "transparent" is set to FALSE, the transparentColor
 * or mask will be cleared.
 *)
  (# value: @boolean;
  enter (#  enter value ... #)
  exit
    (#
    ...
    exit value
    #)
  #);
transparentColor:
(* Specify which color in the pixmap should be transparent.
 * This will normally be the background color in the pixmap.
 * When a transparentColor is specified the "transparent"
 * attribute will be set to TRUE.
 * If the "transparent" is set to FALSE, any transparentColor
 * will be cleared.
 *)
  (# theColor: @color;
  enter
    (#  enter theColor ... #)
  exit
    (#  ... exit theColor #)
  #);
mask:
(* Specify a mask for this(pixmap). A mask is a one−depth
 * pixmap. Only the pixels in this(pixmap) that has
 * corresponding pixel in the mask with the value 1, will
 * be drawn on the screen, when drawing this(pixmap).
 * If a mask is specified the "transparent" attribute will
 * be set to TRUE. If transparent is set to FALSE, any mask
 * will be cleared.
 *)
  (# theMask: ^pixmap;
  enter (#  enter theMask[] ... #)
```

```
      exit
        (#
        ...
        exit
        theMask[]
        #)
      #);
    drawPixmap: (* Draw the pixmap "other" on this(pixmap) *)
      (# other: ^pixmap; from,to: @point; width,height: @integer;
      enter (other[],from,to,width,height)
      ...
      #);
    private: @...;

  #);
textStyle:
(* textStyle is font, size and face. You can use this pattern
 * to communicate stylic changes to layout−text and
 * document−text − or to get information about the dimension of
 * text drawn in a specific textStyle
 *)
  (#
     <<SLOT textStyleLib:Attributes>>;
     name:
     (* models the name of the font of THIS(textStyle). *)
       (# theName: ^text;
       enter
         (#  enter theName[] ... #)
       exit
         (#
         ...
         exit theName[]
         #)
       #);
     size:
       (# value: @integer;
       enter (#  enter value ... #)
       exit (#  ... exit value #)
       #);
     face:
       (# value: @integer;
       enter (#  enter value ... #)
       exit (#  ... exit value #)
       #);
     ascent: integerValue
     (* ascent is the maximum amount of pixels a character
      * drawn in THIS(textStyle) will go above the base line
      *) (#  ... #);
     descent: integerValue
     (* descent is the maximum amount of pixels a character
      * drawn in THIS(textStyle) will go below the base line
      *) (#  ... #);
     leading: integerValue
     (* leading is the vertical distance between the descent
      * of one line and the ascent of the next line
      *) (#  ... #);
     lineHeight: integerValue
     (* the line height (in pixels) is determined by adding
      * the ascent, descent, and leading
      *) (#  ... #);
     maxChWidth: integerValue
     (* the greatest distance the pen will move when a
      * character is drawn
      *) (#  ... #);
     widthOfChar: integerValue
     (* in most fonts the width of the characters
```

```
       * differs. This method returns the width of the character
       * "ch" when drawn in THIS(textStyle)
       *)
        (# ch: @char
        enter ch
        ...
        #);
      widthOfText: integerValue
      (* widthOfText returns the width of the given text
       * string, when drawn in THIS(textStyle), which it
       * calculates by adding the charWidths of all the
       * characters in the string
       *)
        (# str: ^text
        enter str[]
        ...
        #);
      availableSizes:
      (* an INNER is executed for all available sizes in the
       * font of THIS(textStyle)
       *)
        (# thisSize: @integer;
        ...
        #);
      private:
        @...;

  #)
  (* textStyle *)
  ;
color: (* A Color has three components: red, green and blue. *)
  (# <<SLOT colorLib:Attributes>>; red,green,blue: @integer;
  enter (red,green,blue)
  exit (red,green,blue)
  #);
timer:
  (#
     <<SLOT timerLib:Attributes>>;
     once:<
      booleanValue;
     start:
       (# interval: @integer
       enter interval
       ...
       #);
     stop:
       (#
       ...
       #);
     action:< object;
     private: @...;

  #);
clipboard: @
(* models the clipboard, which is used to transport pictures
 * and text between applications
 *)
  (#
     <<SLOT clipBoardLib:Attributes>>;
     hasText: booleanValue
     (* returns true if the contents of the clipBoard is text
      *) (#  ... #);
     textContents:
     (* evaluate the enter−part to set the clipboards
      * text−contents, and evaluate the exit−part to get the
      * clipboards text−contents.  If the clipboard doesn't
```

```
       * contain text, NONE is returned.  You can call hasText,
       * before calling getTextContents to determine if there is
       * text to get
       *)
        (# txt: ^text;
        enter (#  enter txt[] ... #)
        exit (#  ... exit txt[] #)
        #);
      clearContents: (* call this to empty all contents of the clipboard *)
       (#  ... #);

  #)
  (* clipboard *)
  ;
mouse: @ (* models the mouse *)
  (#
     <<SLOT mouseLib:Attributes>>;
     globalPosition:
     (* the global position of the mouse is returned. You
      * can't set the position
      *)
       (# pt: @point;
       ...
       exit pt
       #);
     buttonState: integerValue
     (* the number designating the button, currently pressed
      * down − 0 means 'no button'.  This value depends on the
      * number of buttons on the mouse − Typically 1, 2 or 3.
      *) (#  ... #);
     busyCursor:
     (* A busy cursor is a sign to the user that the
      * application are doing some processing. You will
      * normally use cursors.watch for this purpose. Set
      * busyCursor to none, when done processing.
      *)
       (# theCur: ^cursor;
       enter
         (#  enter theCur[] ... #)
       exit
         (#
         ...
         exit theCur[]
         #)
       #);

  #)
  (* mouse *)
  ;
globalKeyDownAction:
(* GlobalKeyDownAction handles keydown events globally before or after
 * any other handling of the event.
 *
 * Assigning 'true −> done' prevents any other handling of the event.
 *
 * If the action is appended, other handling of the event can not be
 * prevented.
 *)
  (# key: @char; control: @boolean; done: @boolean;
  enter (key,control)
  do false−>done; INNER ;
  exit done
  #);
prependAction:
(* Prepends the global action, which means the action
 * will be executed before any other handling of a keydown
```

```
 * event.
 *)
  (# theAction: ^globalKeyDownAction;
  enter theAction[]
  ...
  #);
appendAction:
(* Appends the action, which means that the action will
 * be executed after any other handling of the event.
 *)
  (# theAction: ^globalKeyDownAction;
  enter theAction[]
  ...
  #);
(* These models different properties of the current system
 * next 5 patterns was formerly in a systempattern
 *)
screenRectangle: (* the rectangle of the main screen. *)
  (# theRectangle: @rectangle;
  ...
  exit theRectangle
  #);
screenRgn:
(* the region defining the screen(s) *)
  (# rgn: ^region;
  ...
  exit rgn[]
  #);
standardTextStyle:
(* the textStyle used by the system to draw menutitles
 * etc.
 *) @textStyle;
beep: (* beeps using the current beep in the system *)
  (#  ... #);
guienvWait:
(* delays the specified number of ticks (1 tick = 1/60
 * sec.)
 *)
  (# ticks: @integer;
  enter ticks
  ...
  #);
transferModes: @
  (#
     copy: (#  exit 0 #);
     invertCopy: (#  exit 1 #);
     erase: (#  exit 2 #);
     andBlend: (#  exit 3 #);
     orBlend: (#  exit 4 #);
     xorBlend: (#  exit 5 #);
     notAndBlend: (#  exit 6 #);
     notOrBlend: (#  exit 7 #);

  #);
textFaces: @
  (#
     <<SLOT textFacesLib:Attributes>>;
     plain: (#  exit 0 #);
     bold: (#  exit 1 #);
     italic: (#  exit 2 #);

  #);
patterns: @ (# black,dkGray,gray,ltGray,white: ^pixmap #);
cursors: @ (# arrow,iBeam,watch,cross,plus,hand: @cursor #);
borderStyles: @
  (#
```

```
         simple: (#  exit 1 #);
         etchedOut: (#  exit 2 #);
         etchedIn: (#  exit 3 #);
         shadowIn: (#  exit 4 #);
         shadowOut: (#  exit 5 #);

      #);
    separatorStyles: @
      (#
         singleLine: (#  exit 1 #);
         doubleLine: (#  exit 2 #);
         singleDashedLine: (#  exit 3 #);
         doubleDashedLine: (#  exit 4 #);
         etchedIn: (#  exit 5 #);
         etchedOut: (#  exit 6 #);

      #);
    windowTypes: @
      (#
         normal: integerValue (#  do 0->value #);
         dialog: integerValue (#  do 1->value #);
         palette: integerValue (#  do 2->value #);
         modelessDialog: integerValue (#  do 3->value #);

      #);
    specKeys: @specialKeys;
    private: @...;
    trace:
    (* For debugging. If doTrace is true, INNER is called. *)
      (#  do (if doTrace then INNER if);  #);
    doTrace: @Boolean;
    (* Additions needed for bifrost *)
    bifrostprivate: @...;
    displaywarnings: @boolean
    (* If displayWarnings is true, various warnings about bifrost
     * errors that are not fatal, but may affect the behaviour, is
     * displayed. Defaults to true.
     *) ;
    warnStream: ^stream
    (* The stream bifrost warnings are put to. Defaults to
     * screen.
     *) ;
    (* Additions needed for systemenv *)
    doSetup:
      (#
      do
         (if not setupDone then
             ...; true->setupDone
         if)
      #);
    setupDone: @Boolean;
    XsystemEnvPresent: @Boolean;
    (* TRUE if this is a XsystemEnv program.  In this case,
     * callbacks are executed by a separate thread as synchronisation
     * via semaphores between x-callbacks and other coroutines would
     * not be possible otherwise. (It could lead to suspend of
     * coroutines with C stackparts.  If TRUE,
     * XsystemEnvHandleCallback should not be NONE.
     *)
    XsystemEnvHandleCallbackP: (# cb: ^Object;  enter cb[] do INNER #);
    XsystemEnvHandleCallback: ^XsystemEnvHandleCallbackP;

...
#)
```

20.9 Guienvactions Interface

```
ORIGIN 'guienv'
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
-- interfaceobjectLib: attributes --
basicEventAction: action
  (# eventType::< theEventhandler.basicEvent;
  do INNER
  #);
mouseEventAction: basicEventAction
  (# eventType::< theEventhandler.mouseEvent;
  do INNER
  #);
mouseDownAction: mouseEventAction
  (# eventType::< theEventhandler.mouseDown
  do INNER
  #);
mouseUpAction: mouseEventAction
  (# eventType::< theEventhandler.mouseUp;
  do INNER
  #);
keyEventAction: basicEventAction
  (# eventType::< theEventhandler.keyEvent;
  do INNER
  #);
keyDownAction: keyEventAction
  (# eventType::< theEventhandler.keyDown;
  do INNER
  #);
refreshAction: basicEventAction
  (# eventType::< theEventhandler.refresh
  do INNER
  #);
activateAction: basicEventAction
  (# eventType::< theEventhandler.activate
  do INNER
  #);
deactivateAction: basicEventAction
  (# eventType::< theEventhandler.deactivate
  do INNER
  #);


-- windowLib: attributes --
aboutToCloseAction: action
  (# eventType::< theEventhandler.aboutToClose;
  do INNER;
  #);

-- menuitemLib: attributes --
selectAction: action
  (# eventType::< theEventhandler.select;
  do INNER
  #);

-- menuLib: attributes --
selectAction: action
  (# eventType::< theEventhandler.select;
  do INNER
  #);
```

```
-- windowitemLib: attributes --
visibleChangedAction: action
  (# eventType::< theEventhandler.visibleChanged
  do INNER
  #);
frameChangedAction: action
  (# eventType::< theEventhandler.frameChanged
  do INNER
  #);
fatherFrameChangedAction: action
  (# eventType::< theEventhandler.fatherFrameChanged
  do INNER
  #);
enabledChangedAction: action
  (# eventType::< theEventhandler.enabledChanged
  do INNER
  #);
enableTargetAction: action
  (# eventType::< theEventhandler.enableTarget
  do INNER
  #);
disableTargetAction: action
  (# eventType::< theEventhandler.disableTarget
  do INNER
  #);
borderVisibleChangedAction: action
  (# eventType::< theEventhandler.borderVisibleChanged;
  do INNER
  #);
borderStyleChangedAction: action
  (# eventType::< theEventhandler.borderStyleChanged;
  do INNER
  #);
theCursorChangedAction: action
  (# eventType::< theEventhandler.theCursorChanged;
  do INNER
  #);
hiliteChangedAction: action
  (# eventType::< theEventhandler.hiliteChanged
  do INNER
  #);

-- separatorLib: attributes --
styleChangedAction: action
  (# eventType::< theEventhandler.styleChanged;
  do INNER
  #);

-- canvasLib: attributes --
childFrameChangedAction: action
  (# eventType::< theEventhandler.childFrameChanged
  do INNER
  #)
```

20.10 Guienvall Interface

```
ORIGIN 'guienv';
INCLUDE 'stddialogs';
INCLUDE 'controls';
INCLUDE 'fields';
INCLUDE 'figureitems';
INCLUDE 'scrolllists';
INCLUDE 'graphmath';
INCLUDE 'graphics';
INCLUDE 'styledtext';
INCLUDE 'guienvactions';
INCLUDE 'controlsactions';
INCLUDE 'fieldsactions';

-- lib: attributes --

empty: (# #)
```

20.11 Guienvsystemenv Interface

```
ORIGIN '~beta/basiclib/basicsystemenv';
MDBODY nti     'private/winnt/guienvntsystemenvbody'
       ppcmac  'private/macintosh/guienvsystemenvmac.bet'
       default 'private/X11/guienvxsystemenvbody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *
 * GUIENVSYSTEMENV
 * ===============
 *
 * Use this fragment as the ORIGIN for concurrent programs
 * using the GUIENV libraries.
 *
 * The program should look something like:
 *
 * ORIGIN 'guienvsystemenv';
 * --- program: descriptor ---
 * systemEnv
 * (# setWindowEnv::< (# do myWindowEnv[] -> theWindowEnv[] #);
 *    myWindowEnv: @guienv (# ... #);
 *    ...
 * #);
 *
 * The 'setWindowEnv' virtual and 'theWindowEnv' reference are
 * declared in basicsystemenv.
 *
 * The guienv instance (myWindowEnv) assigned to theWindowEnv is
 * used for scheduling purposes to allow BETA coroutines to
 * cooperate with the event driven user interface.
 *
 * For concurrency details, see basicsystemenv.
 *)

(* deliberate empty fragment file *)
```

20.12 Scrolllists Interface

```
ORIGIN 'guienv';
LIB_DEF 'guienvscrolllists' '../lib';

BODY 'private/scrolllistsbody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
-- windowLib: attributes --
scrollList: windowItem
  (# <<SLOT scrollListLib: attributes>>;
     numberOfItems: integerValue
       (* returns the number of items in THIS(scrollList). The items
        * in the scrollList are indexed from 1 to size. That is, size
        * is the index to the last item
        *)
       (# do ... #);
     insert:
       (* the specified number of items (numOfItems) are inserted
        * before the specified item (beforeItem). The items in the
        * scrollList are indexed from 1 to value returned by "numOfItems"
        *)
       (# beforeItem, numOfItems: @integer;
       enter (beforeItem, numOfItems)
       do ...
       #);
     prepend:
       (* the specified number of items (numOfItems) are inserted at
        * the beginning of THIS(scrollList). The items in the
        * scrollList are indexed from 1 to value returned by "numOfItems"
        *)
       (# numOfItems: @integer;
       enter numOfItems
       do ...
       #);
     append:
       (* the specified number of items (numOfItems) are inserted at
        * the end of THIS(scrollList). The items in the scrollList are
        * indexed from 1 to value returned by "numOfItems"
        *)
       (# numOfItems: @integer;
       enter numOfItems
       do ...
       #);
     delete:
       (* the specified range of items are deleted from
        * THIS(scrollList).  FirstItem is the index of the first item
        * to be deleted.  The items in the scrollList are indexed from
        * 1 to value returned by "numOfItems"
        *)
       (# firstItem, numOfItems: @integer;
       enter (firstItem, numOfItems)
       do ...
       #);
     deleteFirst:
       (* the specified range of items are deleted from the beginning
        * of THIS(scrollList). The items in the scrollList are indexed
        * from 1 to value returned by "numOfItems"
        *)
       (# numOfItems: @integer;
       enter numOfItems
```

```
  do ...
  #);
deleteLast:
  (* the specified range of items are deleted from the end
   * of THIS(scrollList). The items in the scrollList are indexed
   * from 1 to value returned by "numberOfItems"
   *)
  (# numOfItems: @integer;
  enter numOfItems
  do ...
  #);
itemHeight:
  (* the height in pixels of a Item in THIS(scrollList) is the
   * same for all items.  Evaluate the enter−part to set the
   * height.  Evaluate the exit−part to get the height
   *)
  (# h: @integer;
  enter (# enter h do ... #)
  exit (#  do ... exit h #)
  #);
multipleSelection:<
  (* if multipleSelection is TRUE the user is allowed to select
   * multiple elements in the list at a time.
   *)
  booleanValue;
getItemRectangle:
  (* returns the rectangle occupied by the item specified by the
   * item index "theItem". The rectangle is in terms of the
   * coordinate system of the father of THIS(scrollList)
   *)
  (# theItem: @integer;
     theRectangle: @rectangle;
  enter theItem
  do ...
  exit theRectangle
  #);
selection: @
  (# clear:
       (* deselects all items in THIS(scrollList) *)
       (# do ... #);
     scrollIntoView:
       (* scrolls THIS(scrollList) so the selected item are
        * visible
        *)
       (# do ... #);
     first: integerValue
       (* returns the index to the first selected item in
        * THIS(scrollList). There might be items between the
        * first and the last selected item that are not selected
        *)
       (# do ... #);
     last: integerValue
       (* returns the index to the last selected item in
        * THIS(scrollList). There might be items between the
        * first and the last selected item that are not selected
        *)
       (# do ... #);
     select:
       (* selects the item specified by the item index
        * "theItem".  if extend is TRUE the item is added to the
        * selection, otherwise the selection is first emptied
        *)
       (# theItem: @integer;
          extend: @boolean;
       enter (theItem,extend)
       do ...
```

```
            #);
          deselect:
            (* deselects the item specified by the item index
             * "theItem".  If the item wasn't selected nothing happens
             *)
            (# theItem: @integer;
            enter theItem
            do ...
            #);
          has: booleanValue
            (* returns whether the item specified by the item index
             * "theItem".  is selected
             *)
            (# theItem: @integer
            enter theItem
            do ...
            #);
        #) (* selection *);
      scanSelection:
        (# current: @integer;
        do ...
        #);
      scan:
        (# current: @integer;
        do ...
        #);
      open::< (# create::< (# do ... #);
        do ...
        #);
      close::< (# do ... #);
      eventhandler::<
        (# select: event
             (* Called, when the user selects an item in
              * this(scrollList). `Item' is the index of
              * the item in this(scrollList) and `doubleClick'
              * is true it the item was selected by a double
              * click.
              *)
            (# item: @integer;
               doubleClick: @boolean;
            enter (item, doubleClick)
            do INNER;
            #);
          onSelect:< select;
          onFrameChanged::<
            (# do ... #);
          onRefresh::<
            (# do ... #);
          onMouseDown::<
            (# do ... #);
          onActivate::<
            (# do ... #);
          onDeactivate::<
            (# do ... #);
          onVisibleChanged::<
            (# ... #);
        #);
      private: @...;
  #) (* scrollList *);
textScrollList: scrollList
  (# <<SLOT textScrollListLib: attributes>>;
     setText:
       (* the item specified by the item index "theItem" is set to
        * the text "theText"
        *)
       (# theText: ^text;
```

```
        theItem: @integer;
      enter (theItem,theText[])
      do ...
      #);
    getText:
      (* the text in the item specified by the item index "theItem"
       * is returned
       *)
      (# theText: ^text;
         theItem: @integer;
      enter theItem
      do ...
      exit theText[]
      #);
    style:
      (* the style used to display the item texts in
       * THIS(textScrollList).  Evaluate the enter−part to set the
       * style.  Evaluate the exit−part to get the style
       *)
      (# setTextStyle:
           (# theStyle: ^textStyle;
           enter theStyle[]
           do ...
           #);
         getTextStyle:
           (# theStyle: ^textStyle;
           do ...
           exit theStyle[]
           #);
      enter setTextStyle
      exit getTextStyle
      #);
    open::<
      (# create::< (# do ... #);
      do ...
      #);
    close::< (# do ...; #);
  #)
```

20.13 Stddialogs Interface

```
ORIGIN 'guienv';
LIB_DEF 'guienvstddialogs' '../lib';

BODY 'private/stddialogsbody'
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *
 * The intent of this fragment is that is should contain verious
 * standard dialogs, such as noteUser, alertUser, fileSelectionDialog,
 * etc.
 *)
-- guienvLib: attributes --
dialog:
  (* Dialog is an abstract superpattern for activating
   * modal dialogs. In the INNER of this(dialog) you
   * can assign values to owner and dialogTitle to control
   * these features of the dialog.
   *
   * If the dialogwindow has a titlebar the title is used.
   * If the owner is specified, the dialog is centered
   * inside that window. If owner is NONE, the dialog is centered
   * on the screen.
   *)
  (# owner: ^window;
     title: ^text;
     private: @...;
  enter owner[]
  ...
  #);

messageDialog: dialog
  (* MessageDialog is an abstract superpattern for dialogs with
   * a simple message.
   *
   * The message can be specified by evaluating the enter part, eg.
   *   (NONE, 'You have new mail', 'Mail Dialog')-> noteUser;
   *
   * and in the INNER:
   *   newMailDialog: noteUser
   *    (# do 'You have new mail' -> message[] #);
   *)
  (# message: ^text;
     messageDialogPrivate: @...;
     onClose:< object;
  enter (message[], title[])
  ...
  #);
noteUser: messageDialog
  (* A note user dialog are used for neutral messages.
   * No additional features are defined here.
   *)
  (#
  ...
  #);
alertUser: messageDialog
  (* AlertUsers brings up a simple messagedialog
   * with a warning icon. Use it to warn to user of
   * som dangerous condition.
   *)
  (#
```

```
  ...
  #);

fileDialog: dialog
  (* FileDialog is an abstract superpattern for
   * file selection and file creation (On some platforms
   * these two dialogs are actually the same).
   *
   * The filter is an wildcard like: '*.c'. If filter is none, '*'
   * is used.
   * The path is the a path to the default directory in the dialog.
   * If path is none, the working directory is used.
   * Label is the label for the textfield displaying the current
   * selection.
   * filter, path, label, filename, title may be set in the do−part.
   *)
  (# filter, path, label, fileName, defaultExt: ^text;
     fileDialogPrivate: @...;
  ...
  exit fileName[]
  #);
fileSelectionDialog: fileDialog
  (* brings up a standard file selection dialog *)
  (# ... #);
fileCreationDialog: fileDialog
  (* brings up a standard file creation dialog *)
  (# ... #)
```

20.14 Styledtext Interface

```
ORIGIN '~beta/basiclib/file'
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
-- lib: attributes --
styledText: text(# styleInfo: @integer #)
```

21.1 Buttonadds Interface

```
ORIGIN '../controls';

INCLUDE '../private/controlsbody'

--- buttonlib: attributes ---
labelHeight:
  (# ts: ^textStyle;
  do style->ts[];
  exit ts.lineHeight
  #);
labelWidth:
  (# ts: ^textStyle;
  do style->ts[];
  exit (label->ts.widthOfText)
  #)
```

21.2 ColorConverter Interface

```
ORIGIN '~beta/basiclib/betaenv';
BODY 'private/colorConverterBody'
(*
 * ColorConverter:
 *    This file implements conversion routines between the different
 *    color models.  Currently, three color models are supported:
 *    RGB, CMY and HSV.
 *
 *    The implementation of HSVtoRGB and RGBtoHSV routines
 *    are based on the Bifrost implementation (PaintImpl.bet)
 *
 * Usage:
 *    converter: @colorConverter;
 *    ...
 *    RGBcolor->converter.RGBtoHSV->HSVcolor;
 *    ...
 *    HSVcolor->converter.HSVtoRGB->RGBcolor;
 *
 * You can costumize this converter to convert into RGB and HSV
 * color models with different ranges for the dimensions in the
 * color models by further binding the maxRGB, MaxHue, MaxSat, and
 * maxVal virtuale in colorConverter.  E.g. to use a 256 RGB cube,
 * declare:
 *
 *    converter: @colorConverter(# maxRGB:: (# do 256->value #);
 *
 * Please note, that due to the duality of the RGB and CMY color models,
 * they use the same MaxRGB constant.
 *)
--- LIB: attributes ---

colorConverter:
  (# (* Contants specifying the range for RGB, hue, saturation and value *)
     MaxRGB:< IntegerValue(# do 65535->value; INNER #);
     MaxHue:< IntegerValue(# do   360->value; INNER #);
     MaxSat:< IntegerValue(# do 32768->value; INNER #); (* (2^15) *)
     MaxVal:< IntegerValue(# do 32768->value; INNER #); (* (2^15) *)

     HSVtoRGB:
       (* Convert h in [0, MaxHue], s in [0, MaxSat], V in [0, MaxVal]
        * to RGB-values in [0, MaxRGB].
        * Adapted from algorithm in
        * "Foley and van Dam: Fundamentals of Interactive Computer Graphics"
        *)
       (# H, S, V: @integer;
          R, G, B: @integer;
          convert: @...
       enter (H, S, V)
       do convert
       exit (r, g, b)
       #);
     RGBtoHSV:
       (* Obtain HSV from RGB *)
       (# R, G, B: @integer;
          H, S, V: @integer;
          convert: @...
       enter (r, g, b)
       do convert
       exit (H, S, V)
       #);
     CMYtoRGB:
       (* convert CMY color to RGB color *)
```

```
      (# C, M, Y: @integer;
         R, G, B: @integer;
         max: (* private *) @integer;
      enter (C, M, Y)
      do maxRGB->max; (max-C, max-M, max-Y)->(R, G, B)
      exit (R, G, B)
      #);
   RGBtoCMY:
      (* convert RGB color to CMY color *)
      (# R, G, B: @integer;
         C, M, Y: @integer;
         max: (* private *) @integer;
      enter (R, G, B)
      do maxRGB->max; (max-R, max-G, max-Y)->(C, M, Y)
      exit (C, M, Y)
      #);
#)
```

21.3 ColorDialog Interface

```
ORIGIN '~beta/guienv/guienv';
BODY 'private/colorDialogBody'
-- guienvLib: Attributes --
colorDialog: window
  (* this pattern implements a color selection dialog in which you
   * can specify a color interactively in a dialog window.  You can
   * specify explicit color values in either RGB or HSV color values,
   * select colors from a set of predefined color names, or you can
   * select the color directly in a HSV color space.
   *
   * Interactively, a color is selected as a custom color value by
   * 'dragging' the color from the 'Color' area down into one of the
   * color spots in the 'Custom' group.
   *)
  (# apply:<
       (* this is invoked each time either 'Apply' or 'Close' button
        * are pressed.  If the 'Close' button was pressed, 'closing'
        * is 'true'.  When invoked, 'c' will contain the selected
        * color value
        *)
       (# c: @color; closing: @boolean
       enter closing
       ...
       #);
     setColor:
       (* You can set the custom colors or the selected color by
        * invoking this operation. If 'no'>0, the corresponding custom
        * color will be set.  If no=0, then the selected color will be
        * set.  In both cases, the new color is expected in 'c'.
        *)
       (# no: @integer; c: @color
       enter (no,c)
       ...
       #);
     getColor:
       (* You can get the custom colors or the selected color by
        * invoking this operation. If 'no'>0, the corresponding custom
        * color will be returned in 'c'.  If no=0, then the selected
        * color will be returned in 'c'.
        *)
       (# no: @integer; c: @color
       enter no
       ...
       exit c
       #);
     open::< (# ... #);
     private: @...
  #)
```

21.4 ColorTable Interface

```
ORIGIN '../guienv';
BODY 'private/colorTableBody'

(* This fragment adds color-table facilities to guienv.
 * A 'colorTable' pattern have been added.
 *
 * colorTable
 * ----------
 *
 * This pattern defines facilities for maintaining mappings between
 * textual names of colors, and their numerical representations (in terms
 * of 'color' values). 'colorTable' defines four functions:
 *
 * define: defines the text in 'colorName' to refer to the current
 *   color of 'colorValue'.  If 'colorName' is already defined, it
 *   will be redefined to refer to the color of 'colorValue'.
 *
 * lookup: sets the color values of 'colorValue' to the color values
 *   referred to by 'colorName'.  If 'colorName' is not defined in the
 *   colortable, 'colorValue' will be set to 'black'
 *
 * loadFile: load the color definitions found in the file 'filename' into
 *   the color table.
 *
 *   If 'overwrite' is bound to 'trueObject'
 *   (i.e. 'overwrite::trueObject'), then the new color definitions
 *   will replace any previously existing color definition with the
 *   same color name.
 *
 *   If 'merge' is not bound to 'trueObject' - i.e. not
 *   'merge::trueObject', then clear the color table.
 *
 *   If 'filename' is empty, or cannot be read, the result will be
 *   that all color definitions will be erased, i.e. a way to clear
 *   all existing color definitions is merely by executing:
 *
 *      ct.loadFile; ('ct' assumed to be an instance of 'colorTable')
 *
 * scan: scans the defined colornames in the color table and their
 *   color values.
 *
 * In multi-fragment guienv programs it can be a problem to
 * reach the single colortable instans. This problem can be
 * solved using the object-pool.
 *
 * Main fragment:
 * ct:@colortable;
 * ...
 * ct[]->objectPool.put;
 *
 * Other fragment:
 * ct:^colortable;
 * ...
 * objectPool.get(# type::<colortable; #)->ct[];
 *
 * see basiclib reference manual p 12&23
 *)

--- guienvlib: attributes ---
colorTable:
  (# <<SLOT colortablelib: attributes>>;
     define:
```

```
  (* defines the text in 'colorName' to refer to the current
   * color of 'colorValue'.  If 'colorName' is already defined,
   * it will be redefined to refer to the color of 'colorValue'.
   *)
  (# colorName: ^text; colorValue: @color
  enter (colorName[], colorValue)
  ...
  #);
lookup:
  (* sets the color values of 'colorValue' to the color values
   * referred to by 'colorName'.  If 'colorName' is not defined
   * in the colortable, 'colorValue' will be set to 'black'
   *)
  (# colorName: ^text; colorValue: @color
  enter colorName[]
  ...
  exit colorValue
  #);
load:
  (* reads color definitions from a stream
   * into the color table.
   *
   * If 'overwrite' is bound to 'trueObject'
   * (i.e. 'overwrite::trueObject'), then the new color
   * definitions will replace any previously existing color
   * definition with the same color name.
   *
   * If 'merge' is not bound to 'trueObject' − i.e. not
   * 'merge::trueObject', then clear the color table.
   *)
  (# colorstream: ^stream;
     overwrite:< booleanValue;
     merge:< booleanValue;
  enter colorstream[]
  ...
  #);
loadFile:
  (* load the color definitions found in the file 'filename'
   * into the color table.
   *
   * If 'overwrite' is bound to 'trueObject'
   * (i.e. 'overwrite::trueObject'), then the new color
   * definitions will replace any previously existing color
   * definition with the same color name.
   *
   * If 'merge' is not bound to 'trueObject' − i.e. not
   * 'merge::trueObject', then clear the color table.
   *
   * If 'filename' is empty, or cannot be read, the result will
   * be that all color definitions will be erased, i.e. a way to
   * clear all existing color definitions is merely by executing:
   *
   *      ct.load; ('ct' assumed to be an instance of 'colorTable')
   *)
  (# filename: ^text;
     overwrite:< booleanValue;
     merge:< booleanValue;
  enter filename[]
  ...
  #);
scan:
  (* scans the defined colornames in the color table and their color
   * values.
   *)
  (# currentName: ^text; currentColor: @color;
  ...
```

```
      #);
   private: @...
#)
```

21.5 CursorTable Interface

```
ORIGIN '../guienv';
BODY 'private/cursorTableBody'

(* This fragment adds cursor-table facilities to guienv.
 * A 'cursorTable' pattern have been added.
 *
 * cursorTable
 * -----------
 *
 * This pattern defines facilities for maintaining mappings between
 * textual names of cursors, and their numerical representations (in terms
 * of 'cursor' values). 'cursorTable' defines four functions:
 *
 * define: defines the text in 'cursorName' to refer to the current
 *   cursor of 'cursorValue'.  If 'cursorName' is already defined, it
 *   will be redefined to refer to the cursor of 'cursorValue'.
 *
 * lookup: sets the cursor values of 'cursorValue' to the cursor values
 *   referred to by 'cursorName'.  If 'cursorName' is not defined in the
 *   cursortable, 'cursorValue' will be set to 'black'
 *
 * loadFile: load the cursor definitions found in the file 'filename' into
 *   the cursor table.
 *
 *   If 'overwrite' is bound to 'trueObject'
 *   (i.e. 'overwrite::trueObject'), then the new cursor definitions
 *   will replace any previously existing cursor definition with the
 *   same cursor name.
 *
 *   If 'merge' is not bound to 'trueObject' - i.e. not
 *   'merge::trueObject', then clear the cursor table.
 *
 *   If 'filename' is empty, or cannot be read, the result will be
 *   that all cursor definitions will be erased, i.e. a way to clear
 *   all existing cursor definitions is merely by executing:
 *
 *      ct.loadFile; ('ct' assumed to be an instance of 'cursorTable')
 *
 * scan: scans the defined cursornames in the cursor table and their
 *   cursor values.
 *
 * In multi-fragment guienv programs it can be a problem to
 * reach the single cursortable instans. This problem can be
 * solved using the object-pool.
 *
 * Main fragment:
 * ct:@cursortable;
 * ...
 * ct[]->objectPool.put;
 *
 * Other fragment:
 * ct:^cursortable;
 * ...
 * objectPool.get(# type::<cursortable; #)->ct[];
 *
 * see basiclib reference manual p 12&23
 *)

---cursorlib: attributes ---
     load:
       (# x11ID:< (# ID: @integer do INNER exit ID #);
          macID:< (# ID: @integer do INNER exit ID #);
```

```
         ntiID:< (# ID: @integer do INNER exit ID #);
         ntiNAME:< (# ID: ^text do INNER exit ID[] #);
      ...
      #);

--- guienvlib: attributes ---
cursorTable:
  (# <<SLOT cursortablelib: attributes>>;
     init:< (* is called automatically to enable automatic load of cursors *)
       (# do INNER init #);
     create:
       (# x11ID:< (# ID: @integer do INNER exit ID #);
          macID:< (# ID: @integer do INNER exit ID #);
          ntiID:< (# ID: @integer do INNER exit ID #);
          ntiNAME:< (# ID: ^text do INNER exit ID[] #);
          cursorName: ^text;
          c: ^(*private*)cursor;
       enter cursorName[]
       ...
       #);
     define:
       (* defines the text in 'cursorName' to refer to the current
        * cursor of 'cursorRef'.  If 'cursorName' is already defined,
        * it will be redefined to refer to the cursor of 'cursorRef'.
        *)
       (# cursorName: ^text; cursorRef: ^cursor
       enter (cursorName[], cursorRef[])
       ...
       #);
     lookup:
       (* sets 'cursorRef' to refer to the cursor
        * referred to by 'cursorName'.  If 'cursorName' is not defined
        * in the cursortable, 'cursorRef' will be set to 'arrow'
        *)
       (# cursorName: ^text; cursorRef: ^cursor
       enter cursorName[]
       ...
       exit cursorRef[]
       #);
     load:
       (* reads cursor definitions from a stream
        * into the cursor table.
        *
        * If 'overwrite' is bound to 'trueObject'
        * (i.e. 'overwrite::trueObject'), then the new cursor
        * definitions will replace any previously existing cursor
        * definition with the same cursor name.
        *
        * If 'merge' is not bound to 'trueObject' − i.e. not
        * 'merge::trueObject', then clear the cursor table.
        *)
       (# cursorstream: ^stream;
          overwrite:< booleanValue;
          merge:< booleanValue;
       enter cursorstream[]
       ...
       #);
     loadFile:
       (* load the cursor definitions found in the file 'filename'
        * into the cursor table.
        *
        * If 'overwrite' is bound to 'trueObject'
        * (i.e. 'overwrite::trueObject'), then the new cursor
        * definitions will replace any previously existing cursor
        * definition with the same cursor name.
        *
```

```
       * If 'merge' is not bound to 'trueObject' − i.e. not
       * 'merge::trueObject', then clear the cursor table.
       *
       * If 'filename' is empty, or cannot be read, the result will
       * be that all cursor definitions will be erased, i.e. a way to
       * clear all existing cursor definitions is merely by executing:
       *
       *      ct.load; ('ct' assumed to be an instance of 'cursorTable')
       *)
      (# filename: ^text;
         overwrite:< booleanValue;
         merge:< booleanValue;
      enter filename[]
      ...
      #);
    scan:
      (* scans the defined cursornames in the cursor table and their cursor
       * values.
       *)
      (# currentName: ^text; currentCursor: ^cursor;
      ...
      #);
    private: @...
#)
```

21.6 Decorator Interface

```
ORIGIN '../guienv';
LIB_DEF 'guienvutilsdecorator' '../../lib';
BODY 'private/decoratorbody';

-- windowLib: attributes --

decorator: canvas
  (# <<SLOT decoratorLib: attributes>>;
     contentsType:< windowItem;
     contents: @contentsType;
     open::<(#
       ...
       #);
     eventHandler::<
       (# onFrameChanged::<
            (# ... #);
          onChildFrameChanged::<
            (# ... #);
       #);
     private: @...;
  #)
```

21.7 Defaultstyle Interface

```
ORIGIN '../guienv';
BODY 'private/defaultstylebody';

-- GUIenvLib: attributes --

defaultStyle:
  (# style: ^textStyle;
  ...
  exit style[]
  #)
```

21.8 Designsupport Interface

```
ORIGIN '../guienv';
LIB_DEF 'guienvutilsdesignsup' '../../lib';
BODY 'private/designsupportbody';

-- guienvLib: attributes --

designSetup:
  (#
  ...
  #)
```

21.9 Dialogfield Interface

```
ORIGIN '../guienv';
LIB_DEF 'guienvutilsdialogfld' '../../lib';
BODY 'private/dialogfieldbody';

INCLUDE '../figureitems';
INCLUDE '../fields';

-- WindowLib: Attributes --

DialogField: EditText
  (#
     Selection:
       (# start,end: @Integer;
       enter (# enter (start,end) do ... #)
       exit (# do ... exit (start,end) #)
       #);
     Length:
       (# theLength: @Integer;
       do ...;
       exit theLength
       #);
     MaxChar:
       (# maxCh: @Integer;
       enter (# enter maxCh do ...#)
       exit (# do ... exit maxCh #)
       #);

     EventHandler::<(# beforeChange: event
            (# position,length: @integer;
               allow:@boolean;
            enter (position,length)
            do true -> allow;
               INNER;
            exit allow
            #);
          onBeforeChange:< beforeChange (# do ...;
                          #);
          onKeyDown::<
            (# do ...;
            #);
          onEnableTarget::< (# do ...; #);
       #);
     Open::< (# do ... #);
     <<SLOT DialogFieldLib: Attributes>>;
     Private: @...;
  #);

NumberField: DialogField
  (# IntegerContents:
       (# int: @Integer;
       enter (# enter int do ... #)
       exit (# do ... exit int #)
       #);
     EventHandler::<
       (# onBeforeChange::< (# do ...; #);
       #);
  #);

AlphaNumField: DialogField
  (# EventHandler::<
       (# onBeforeChange::< (# do ...; #);
       #);
```

```
  #);

LabeledCanvas: Canvas
  (# Label: @StaticText(# #);
     Sep: @Separator(# #);
     Open::< (# do ... #);
     <<SLOT LabeledCanvasLib: Attributes>>;
  #)
```

21.10 Graphicsadds Interface

```
ORIGIN '../graphics';
LIB_ITEM 'guienv';
BODY 'private/graphicsaddsbody';
INCLUDE 'rasteradds';

-- graphicsLib: attributes --
clipRectangle:
  (# r: @rectangle;
  enter r
  ...
  #);
isInClip: booleanValue
  (# r: @rectangle;
  enter r
  ...
  #);
drawIconResource:
  (# r: ^iconResource;
     h,v: @integer;
  enter (r[],h,v)
  ...
  #)
```

21.11 Group Interface

```
ORIGIN '../guienv';
LIB_DEF 'guienvutilsgroup' '../../lib';
BODY 'private/groupbody';

-- windowLib: attributes --

group: canvas
  (# <<SLOT groupLib: attributes>>;
     label:
       (# theLabel: ^text;
       enter (# enter theLabel[] ... #)
       exit (#  ... exit theLabel[] #)
       #);
     open::<(# create::< (# ... #)
       ...
       #);
     eventHandler::<
       (#
          onRefresh::<
            (# ... #);
       #);
     private: @...;
  #)
```

21.12 Guienvadds Interface

```
ORIGIN '../guienv';
LIB_DEF 'guienvadds' '../../lib';
BODY 'private/guienvaddsbody';
-- windowItemLib: attributes --


defineRect:
  (* Let the use drag the rectangle.
   * doLeft,doTop,doRight,doBottom specifies which
   * sides of the rectangle should follow the mouse.
   *)
  (# r: @rectangle;
     doLeft,doTop,doRight,doBottom: @boolean;
  enter (r,doLeft,doTop,doRight,doBottom)
  ...
  exit r
  #);
bringToFront:
  (* Bring this(windowItem) to the front of its brothers *)
  (#
  ...
  #);
bringBack:
  (#
  ...
  #);

preferredSize:
  (* If this(windowItem) can accept the suggested width and height
   * these wille be returned otherwise some other values are returned.
   *)
  (# suggestedWidth, suggestedHeight: @integer;
     suggest: @boolean;
     preferredWidth, preferredHeight: @integer;
  enter (#
        enter (suggestedWidth, suggestedHeight)
        do true -> suggest;
        #)
  ...
  exit (preferredWidth, preferredHeight)
  #);
drawShadows:
  (# r: @rectangle;
     type: @integer;
  enter (r, type)
  ...
  #);

delegateMouseEvents:
  (# value: @boolean;
  enter value
  ...
  #);


-- windowLib: attributes --

translate:
  (# from, to: ^windowItem;
     p: @point;
     result: @point;
  enter (p, from[], to[])
```

```
  ...
  exit result
  #);

launchFile:
  (# fileName: ^text;
  enter fileName[]
  do ...
  #);

setWindowIcon:
  (# id: @integer;
  enter id
  ...
  #);

setWindowIconPixmap:
  (# pm:^pixmap
  enter pm[]
  ...
  #);

loadWindowIcon:
  (# filename: ^text;
  enter filename[]
  ...
  #);

setApplIcon:
  (# id: @integer;
  enter id
  ...
  #);

loadMouseCursor:
  (# filename: ^text;
  enter filename[]
     ...
  #);

setMouseCursor:
  (# symbolConstant: @integer;
  enter symbolConstant
     ...
  #)
```

21.13 Guienvstuff Interface

```
ORIGIN '../controls';
LIB_DEF 'guienvutilstuff' '../../lib';
BODY 'private/guienvstuffbody';

-- buttonLib: attributes --

center: (# exit 1 #);
left: (# exit 2 #);
right: (# exit 3 #);

alignment:
  (# value: @integer
  enter value
  ...
  #)
```

21.14 Heapview Interface

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/utils/groupbox';
INCLUDE '../graphics';
INCLUDE '~beta/sysutils/heapinfo';
INCLUDE '~beta/basiclib/numberio';
-- windowLib: Attributes --
heapview: canvas
  (#
     onUpdate:<Object;
     ou:@onUpdate;
     t:@timer
       (#
          action::<(# do
               ou; (* onUpdate; *)
               true->Ioacanvas.drawingarea.update;
               true->aoacanvas.drawingarea.update;
               true->CBFAcanvas.drawingarea.update;
            #);
       #);

     open::<
       (#
       do
          (360,130)->size;
          IOAcanvas.open;
          AOAcanvas.open;
          CBFAcanvas.open;
          50->t.start;
          INNER
       #);

     graphview: groupbox
     (# solution:(# exit 50#);
        priv:@
          (# data:[solution]@integer;
             max:@integer;
          #);

        name:<(# t:^text; do INNER; exit t[] #);
        nm:@name;
        value:<IntegerObject;
        val:@value;
        drawingArea:@canvas
          (# draw:@
               graphics
               (# dx:@real;
                  dy:@real;
                  imax:@max;
                  width,height:@integer;
                  mt:@moveto;
                  dt:@drawto;
               do this(canvas).size->(width,height);
                  (for i:solution-1 repeat
                       priv.data[i+1]->priv.data[i];
                  for);
                  val (*value*) ->priv.data[solution];

                  (priv.max,priv.data[solution])->imax->priv.max;

                  width/solution->dx;
                  priv.max/height->dy;
```

```
            (dx,height-priv.data[1]/dy)->mt;
            (for i:solution repeat
                 (i*dx,height-priv.data[i]/dy)->dt;
            for);
         #);
        eventhandler::
         (#
            onrefresh::
              (# cptext:^text;
              do &text[]->cptext[];
                 draw;
                 nm->cptext.puttext;
                 ':'->cptext.put;
                 priv.data[solution]->cptext.putint;
                 cptext[]->title;
              #);
         #);
        open::
         (# w,h:@integer;
         do father.size->(w,h);
            (w-5,h-10)->size;
            (3,14)->position;
            true->bindleft;
            true->bindright;
            1->priv.max;
         #);
     #);
   open::<
     (#
     do name->title;
        drawingArea.open;
        INNER
     #);
#);

IOAcanvas: @graphview
  (#
     Value::
       (# do 5->getHeapInfo->value; #);
     name::(# do 'IOA'->t[];#);
     open::<
       (#
       do
          (110,120)->size;
          (5,5)->position;
          INNER
       #)
  #);
AOAcanvas: @graphview
  (#
     Value::
       (# do 35->getHeapInfo->value;
          value-(36->getHeapInfo)->value;
       #);
     name::(# do 'AOA'->t[];#);
     open::<
       (#
       do
          (110,120)->size;
          (125,5)->position;
          INNER
       #)
  #);
CBFAcanvas: @graphview
  (#
     Value::
```

```
          (# do 15->getHeapInfo->value; #);
        name::(# do 'CBFA'->t[];#);
        open::<
          (#
          do
             (110,120)->size;
             (245,5)->position;
             INNER
          #)
    #)
#)
```

21.15 Iconname Interface

```
ORIGIN '../controls';
LIB_DEF 'guienvutilsiconname' '../../lib';
BODY 'private/iconnamebody';

-- iconButtonLib: attributes --

iconName:
  (# value: ^text;
  enter (# enter value[] ... #)
  exit (# ... #)
  #)
```

21.16 InterfaceObjectAdds Interface

```
ORIGIN '../guienv';
LIB_DEF 'guienvutilsintobjads' '../../lib';
BODY 'private/interfaceObjectAddsBody';

-- interfaceObjectLib: attributes --
isOpen:
  (# val:@boolean;
     ...
  exit val
  #)
```

21.17 Labelled Interface

```
ORIGIN '~beta/guienv/guienv';
LIB_DEF 'guienvutilslabelled' '../../lib';

INCLUDE '~beta/guienv/utils/group';
INCLUDE '~beta/guienv/utils/private/groupbody';

-- windowLib: attributes --

labelled: group
  (#
     <<SLOT labelledLib:attributes>>;

     contentsType:< windowItem;
     contents: @contentsType;

     labelHeight:
       (# ts: ^textStyle;
       do private.theLabel.style->ts[];
       exit ts.lineHeight+3
       #);

     innerFrame:
       (# p: @Point; fr: @rectangle;
       do (3,labelHeight)->fr.topleft;
          size->p; (3,3)->p.subtract; p->fr.bottomright;
       exit fr
       #);

     open::<(# openContents:<
            (# doneInInner: @Boolean;
            do INNER;
               (if not doneInInner then
                   contents.open;
               if);
            #);
       do INNER;
          openContents;
          innerFrame->contents.frame;
          TRUE->contents.bindRight->contents.bindBottom;
       #);
  #)
```

21.18 Listview Interface

```
ORIGIN '../scrolllists';
LIB_DEF 'guienvutilslistview' '../../lib';
BODY 'private/listviewbody';

-- windowLib: attributes --


listView: textScrollList
  (* A textScrollList where each textline in the list corresponds to
   * a unique listItem object.
   *)
  (# <<SLOT listViewLib: attributes>>;
     listItem:
       (* A listItem object corresponds to a unique line in the List
        * The name if this(listItem) i the text in the corresponding
        * line. When the user selects on item by clicking, onSelect is
        * called.
        *)
       (# <<SLOT listItemLib: attributes>>;
          name:
            (* The name is the text in the line corresponding to
             * this(listItem).
             *)
            (# theName: ^text;
            enter (# enter theName[] ... #)
            exit (# ... exit theName[] #)
            #);
          position: integerValue
            (* The the item number of this(listItem) in the range
             * from 1 to numberOfItems.
             *)
            (# ... #);
          init:<
            (* Inserts this(listItem) in the list before
             * `beforeItem', if `beforeItem' is none this(listItem) is
             * appended. Init *must* be the first operation on this(listItem).
             *)
            (# beforeItem: ^listItem;
            enter beforeItem[]
            ...
            #);
          delete:<
            (* Delete this(listItem) from this(listView). *)
            (# ... #);
          onSelect:<
            (* Called when this(listItem) is selected by the user or
             * the programmer. If the user has doubleClicked
             * this(listItem) the `doubleClick' flag is true.
             *)
            (# doubleClick: @boolean;
            enter doubleClick
            do INNER;
            #);
          select:
            (* Select this(listItem). If singeSelection is true, all
             * other items are deSelected.
             *)
            (# ... #);
          deSelect:
            (* Call this to deSelect this(listItem). *)
            (# ... #);
          private: @...;
```

```
      #);
    listItems: @list
      (# element::< listItem #);
    open::<(# ... #);
    close::<
      (# ... #);
    eventHandler::<
      (# onMouseUp::<
           (# ... #);
      #);
    private: @...;
#)
```

21.19 Navigationkeys Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'guienvutilsnavkey' '../../lib';
BODY 'private/navigationkeysbody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)

-- lib: attributes --
leftArrow:
  (# theChar: @char; ... exit theChar #);
rightArrow:
  (# theChar: @char; ... exit theChar #);
upArrow:
  (# theChar: @char; ... exit theChar #);
downArrow:
  (# theChar: @char; ... exit theChar #)
```

21.20 Obguienvadds Interface

```
ORIGIN '~beta/guienv/guienv';
LIB_DEF 'guienvutilsobguiadds' '../../lib';
BODY 'private/obguienvaddsbody';
BODY 'private/obrubberbody';
-- windowLib: Attributes --
wriggle:
  (# pos: @Point; reallyWriggle: @boolean (* nobody wants it now *)
  enter reallyWriggle
  do
     bringToFront;
     (if not reallyWriggle then leave wriggle if);
     position->pos;
     (- 10,0)->pos.add;
     pos->position;
     TRUE->update;
     (for i: 10 repeat
       (if i mod 2 = 1 then
           (20-(2*i),0)->pos.add; pos->position; TRUE->update;
        else
           (- 20+(2*i),0)->pos.add; pos->position; TRUE->update;
       if);

     for);

  #);


-- windowItemLib: Attributes --
(* RESIZERELATIVE
 * ==============
 *
 * Changes the frame of a child windowItem in response, and relative
 * to, a change in the frame of the father frame, as reported by
 * eventHandler.onFatherFrameChanged *)
resizeRelative:
  (# theEvent: ^theEventHandler.onFatherFrameChanged;
  enter theEvent[]
  do ...
  #);
onFatherFrameResizeRelative: action
  (#
     eventType::
      theEventHandler.fatherFrameChanged;

  do theEvent[]->resizeRelative;
  #);
installResizeRelativeAction:
  (#
  do
     &onFatherFrameResizeRelative[]
       ->appendAction;

  #);
itemPos:
(* Returns the position of this windowItem in the coordinatsystem
 * of the surrounding window. *)
  (# pos: @Point;  ... exit pos #);
FrameDrawer:
(* If UseRunningOutline is TRUE, FrameDrawer draws a
 * running outline around THIS(windowItem) until kill is called.
 * Otherwise a black outline is drawn and kept refreshed until kill
 * is called.
```

```
 *)
  (#
     <<SLOT FrameDrawerLib:Attributes>>;
     kill:
       (# doupdate: @Boolean;
       enter doupdate
       ...
       #);
     fdprivate: @...;
     UseRunningOutline: @Boolean;

  enter UseRunningOutline
  ...
  #);
mydrag:
(* Differs from drag by drawing without clipping the children, and
 * by taking a start point from which mousemoves should be handled
 * relatively. Furthermore, the frame of this windowItem is not set,
 * but the resulting frame is returned. *)
  (# startpos: @Point; newFrame: @rectangle;
  enter startpos
  ...
  exit newFrame
  #);


-- canvasLib: Attributes --
getOutline:
(* Lets the user define a default rectangle by dragging and resizing
 * an outline. When an outline has been defined, done is called.
 * When the rectangle is not being resized, it is presented with an
 * illusion of a running outline.
 *)
  (#
     done:< (# result: @rectangle;  enter result do INNER #);
     default: @rectangle;

  enter default
  do ...
  #);
defineRect:
(* Used to resize window rectangles. *)
  (# theRect: @rectangle; followWhenOutside: @Boolean;
  enter (theRect,followWhenOutside)
  do ...
  exit theRect
  #);
simpleDefineRect:
(* Used to define a simple rectangle, starting in the point entered. *)
  (# p: @Point; theRect: @rectangle;
  enter p
  do ...
  exit theRect
  #);
browserDrawRect:
(* Draws (or clears) rectangle. *)
  (# theRect: @rectangle;
  enter theRect
  ...
  #);


-- lib: Attributes --
actionList: List
  (#
     element::  (# action: ##Object #);
```

```
      appendAction:
        (# new: ^element; action: ##Object;
        enter action##
        do &element[]−>new[]; action##−>new.action##; new[]−>append;
        #);

  #);
actionedMenuItemAction:
  (# paramType:< Object; param: ^paramType; checked: @Boolean;
  enter param[]
  do INNER
  exit checked
  #);
actionedMenuItemOnStatus:
  (# paramType:< Object; param: ^paramType; value: @Boolean
  enter param[]
  do INNER
  exit value
  #);


−− menuLib: Attributes −−
actionedMenuItem: menuItem
  (#
     onSelectAction: ##actionedMenuItemAction;
     onStatusAction: ##actionedMenuItemOnStatus;
     paramType:< Object;
     param: ^paramType;
     open::<
       (# itemname: ^Text;
       enter (onSelectAction##,onStatusAction##,param[],itemname[])
       do THIS(actionedMenuItem)[]−>append; itemname[]−>name; INNER ;
       #);
     eventHandler::<
       (#
          onSelect::<
            (#
            do
               (if onSelectAction## <> none then
                   param[]−>onSelectAction−>checked
               if);
               INNER ;

            #);
          onStatus::<
            (#
            do
               (if onStatusAction## <> none then
                   param[]−>onStatusAction−>value
               if);
               INNER ;

            #);

       #);

  #)
```

21.21 Pane Interface

```
ORIGIN '../guienv';
LIB_DEF 'guienvutilspane' '../../lib';
BODY 'private/panebody';
--- windowLib:attributes ---
pane: Canvas
  (*
   * Initialization:
   *   1. Call open.
   *   2. From within INNER open, call appendMember repeatedly until all
   *      initial members have been appended.
   *   3. From within INNER open, call appendsDone.
   *
   * Extension:
   *   1. Call appendMember repeatedly until all new members have
   *      been appended.
   *   2. Call appendsDone.
   *
   * Deletion:
   *   1. Call deleteMember.
   *
   * If the application allows interactive change of the size of the Pane,
   * the members are automatically extended, since correct operation of
   * the pane depends on the assumption that members fill out the
   * pane. DO NOT INTERFERE WITH AUTOMATIC MEMBER RESIZING! E.g., do
   * not bind members, and do not install actions that resize members. This is
   * taken care of by pane.
   *)
  (# <<SLOT paneLib:attributes>>;

     open::<
       (* verticalStacking: Whether members are stacked vert. or horiz.
        * minsize: Minimal height (or width) of members after interaction.
        * panewidth: Width of the separators between members. Minimum is 2. *)
       (# setDefaults:<
            (* May be used to set default values for verticalStacking,
             * minsize, and panewidth. *)
            (# do INNER #);
          verticalStacking: @Boolean;
          minsize, panewidth: @Integer;
       enter (verticalStacking,minsize,panewidth)
       ...
       #);

     appendMember:
       (# member: ^Canvas;
       enter member[]
       ...
       #);
     appendsDone: (# ... #);

     fixedSize:
       (* Will not respond to automatic resizing in vertical or
        * horizontal direction (depending on the stacking order of
        * this pane
        *)
       (# member: ^Canvas;
       enter member[]
       ...
       #);

     deleteMember:
       (# member: ^Canvas;
```

```
      enter member[]
      ...
      #);

    zoomMember:
      (# member: ^Canvas;
      enter member[]
      ...
      #);

    unzoom:
      (#
      ...
      #);

    onMemberResize:< IntegerValue
      (* If onMemberResize=paneFixed (default), then dragging a
       * separator will resize both of its surrounding members, and keep the
       * size of the pane fixed.
       * If onMemberResize=paneResizable, then, in addition, dragging a
       * separator with the Shift key down will resize the member immediately
       * before the dragged separator, and keep the rest of the member
       * sizes unchanged. As a result, the pane itself is resized. *);
    paneFixed: IntegerValue (# do 0->value #);
    paneResizable: IntegerValue (# do 1->value #);

    fitToFather:
      (* Changes the size of this pane to fit the size of its
       * father frame.
       * If fitHoriz is TRUE, the pane is fitted horizontally.
       * Default of fitHoriz is verticalStacking.
       * If fitVert is TRUE, the pane is fitted vertically.
       * Default of fitVert is not verticalStacking.
       *)
      (# fitHoriz:< BooleanObject
           (# do ...; INNER #);
         fitVert:< BooleanObject
           (# do ...; INNER #);
      do ...
      #);
    fatherFrame:<
      (* Further bind fatherFrame in case "father.frame" is not the
       * frame to be fitted against.
       *)
      (# fr: @rectangle;
      do father.frame->fr;  INNER
      exit fr
      #);

    dppriv: @...;

    eventHandler::<(# onFrameChanged::<
           (* Resizes members to maintain invariant that the pane
            * is filled by its members. *)
           (# do ... #);
      #);
#)
```

21.22 PromptForArgs Interface

```
ORIGIN 'prompts';
LIB_DEF 'guienvutilspromptarg' '../../lib';
BODY 'private/promptForArgsBody';

--GuienvLib: attributes--

PromptForArgs: PromptForText
  (* Special PromptForText used to read line of arguments *)
  (# <<SLOT PromptForArgsLib: attributes>>;
     argv: [0]^text (* The arguments contained in the string typed by the user *);
     argc: @integer;
     validate::<(#
       do ...
       #);
  #)
```

21.23 Prompts Interface

```
ORIGIN '../guienv';
LIB_DEF 'guienvutilspromp' '../../lib';
BODY 'private/promptsbody';

--GuienvLib: attributes--

(* Standard text dialogs for guienv.
 * Known problems:
 *     1. Buttons should be highlighted when invoked via <return>,
 *        <esc> etc.
 *     2. Patterns should be specializations of special dialog pattern,
 *        which would make the window manager treat the prompts as dialogs.
 *)

Prompt: window
  (* Dialog with a message text and an OK and Cancel button.
   * When typing <Return> in the text fiels, the OK button
   * is invoked, and when typing <Escape> the cancel button
   * is invoked.
   *)
  (# <<SLOT prompt: attributes>>;
     menubarVisible:: (# do false -> value #);
     type:: (# do windowTypes.dialog->value #);
     ok:< Object
       (* Called when the OK button is clicked *);
     notok:< Object
       (* Called when the 'No' button is pressed (only promptForBoolean )*);
     cancel:< object
       (* Called when the Cancel button is clicked *);
     extra_hspace:< integerValue
       (* Extra horizontal space to put to the right of the fields *);
     extra_vspace:< integerValue
       (* Extra vertical space to put between the fields and the buttons *);
     popup:<
       (* Used to pop up the dialog and wait for the text to be typed *)
       (# titleText, msgText: ^text;
          father: ^window;
       enter (father[], titleText[], msgText[])
       ...
       #);
     okLabel:< (# value: ^text do 'OK'->value[]; INNER exit value[] #);
     notokLabel:<
       (# value: ^text do 'NOT OK'->value[]; INNER exit value[] #);
     cancelLabel:<
       (# value: ^text do 'Cancel'->value[]; INNER exit value[] #);
     open::< (# ... #);
     private: @...;
  enter popup
  #);

PromptForText: Prompt
  (* Prompt with a field to enter a text added. *)
  (# <<SLOT promptForTextLib: attributes>>;
     usertext: ^text; (* The text entered by user *)
     linesInUsertext:< (* number of lines in the text field for user input *)
       integerValue(# do 1->value #);
     ok::<
       (# ... #);
     validate:< BooleanValue
       (* Called when the OK button is pressed. THIS(Prompt) is only
        * dismissed (and INNER in the ok virtual called) if validate returns
        * true.
```

```
        *)
       (# hiliteOnError: @boolean
            (* Hilite text on error. Default true *);
       ...
       #);
     popup::<
       (# defaultText: ^text;
       enter defaultText[]
       ...
       #);
     open::< (# ... #);
     textprivate: @...;
  #);

PromptForInteger: PromptForText
  (* Special PromptForText that will only accept integer input *)
  (# <<SLOT promptForIntegerLib: attributes>>;
     userinteger: @integer (* The integer typed by the user *);
     validate::<
       (# ... #)
  #);

promptForBoolean: Prompt
  (* Special Prompt where the OK buttons has label 'Yes' and
   * there is an extra button with label 'No'.
   *)
  (# ok::<
       (# ... #);
     notok::< (* Called when the 'No' button is pressed *)
       (# ... #);
     okLabel::<
       (# do 'Yes'->value[]; INNER #);
     notokLabel::<
       (# do 'No'->value[]; INNER #);
     cancelLabel::<
       (# do 'Cancel'->value[]; INNER #);
     open::<
       (# ... #)
  #)
```

21.24 Rasteradds Interface

```
ORIGIN '../guienv';
LIB_ITEM 'guienv';
BODY 'private/rasteraddsbody';
(* INCLUDE '../raster'; *)

-- guienvlib: attributes --
iconResource: pixmap
  (# <<SLOT pixmapLib: attributes>>;
     loadById:
       (# id: @integer;
       enter id
       do ...
       #);
  #);

bitmapResource: pixmap
  (# <<SLOT pixmapLib2: attributes>>;
     loadById:
       (# id: @integer;
          success: @boolean;
       enter id
       ...
       exit success
       #);
     loadByName:
       (# name: ^text;
          success: @boolean;
       enter name[]
       ...
       exit success
       #);
  #)
```

21.25 Scrolleradds Interface

```
ORIGIN '../fields';
LIB_DEF 'guienvutilsscrolladds' '../../lib';
BODY 'private/scrolleraddsbody';

-- scrollerLib: attributes --

theScroll:
  (# h, v: @integer;
  enter (# enter (h, v) ... #)
  exit (# ... exit (h, v) #)
  #);

scrollIntoView:
  (# theWindowItem: ^windowItem;
  enter theWindowItem[]
  ...
  #);

viewSize:
  (# width, height: @integer;
  ...
  exit (width, height)
  #);

noTabbingToScrollbars:
  (#
  ...
  #);

contentsIsTarget:
  (# value: @boolean;
  enter value
  ...
  #)
```

21.26 Simplemenu Interface

```
ORIGIN '../guienv';
LIB_DEF 'guienvutilssimplemenu' '../../lib';
BODY 'private/simplemenubody';
-- MenuBarLib: Attributes --
simplemenu: menu
(* Special menu type with interdependant check marks, some other
 * extra features, and a few code saving patterns, that allow
 * for simple specification of a menu.
 *
 * Example:
 *
 *  FileMenu: @simplemenu
 *    (# inew:    @item(# onSelect:: (# do NewFile    #)#);
 *       iopen:   @item(# onSelect:: (# do OpenFile   #)#);
 *       iclose:  @item(# onSelect:: (# do Iconify    #)#);
 *       isave:   @item(# onSelect:: (# do SaveFile   #)#);
 *       isaveas: @item(# onSelect:: (# do SaveFileAs #)#);
 *       iexport: @item(# onSelect:: (# do Export     #)#);
 *       iprint:  @item(# onSelect:: (# do Print      #)#);
 *       iquit:   @item(# onSelect:: (# do Quit       #)#);
 *
 *       open::
 *         (#
 *         do ('New...',    'n') -> inew.newkey;  false -> inew.enabled;
 *            ('Open...',   'o') -> iopen.newkey; false -> iopen.enabled;
 *            ('Close',     'w') -> iclose.newkey;
 *            newseparator;
 *            ('Save',      's') -> isave.newkey; false -> isave.enabled;
 *            ('Save As...')     -> isaveas.new;  false -> isaveas.enabled;
 *            newseparator;
 *            ('Print...',  'p') -> iprint.newkey;
 *            ('Export...', 'e') -> iexport.newkey;
 *            newseparator;
 *            ('Quit',      'q') -> iquit.newkey;
 *         #)
 *     #);
 *
 *  ShapeMenu: @simplemenu
 *    (# ievenodd: @radioitem
 *         (# onSelect:: (# do EvenOddRule->SetFillRule #)#);
 *       iwinding: @radioitem
 *         (# onSelect:: (# do WindingRule->SetFillRule #)#);
 *       open::
 *         (#
 *         do 'Even-Odd Fillrule'->ievenodd.new;
 *            'Winding Fillrule'->iwinding.new;
 *         #);
 *    #);
 *)
  (#
     item: menuitem
       (#
          enabled: @boolean
          (* Set to true or false to enable or disable THIS(item) *) ;
          onSelect:< object (* Shortcut for eventhandler.onSelect *) ;
          onStatus:< booleanValue
            (* Shortcut for eventhandler.onStatus *)
            (# do enabled->value; INNER #);
          new: (* Open and append THIS(Item) with label itemtext *)
            (# itemtext: ^text;
            enter itemtext[]
            ...
```

```
          #);
        newkey: new
        (* Also add key quivalent *)
          (# equiv: @char
          enter equiv
          ...
          #);
        eventhandler::< (#
             onSelect::<
               (#
               ...
               #);
             onStatus::<  (#  ... #);

          #);

    #);
  noOfRadioGroups:< integervalue
  (* The number of independant groups of radioitems.
   * Default 1.
   *) (#  ... #);
  radioitem: item
  (* Special item, that is un−checked when another radioitem
   * in the radiogroup indicated by "checkgroup" is selected.
   *)
    (#
       checkgroup: @integer
       (* The group of items THIS(item) is dependant on. default 1. *) ;
       open::<  (#  ... #);
       onSelect::<
         (#
         ...
         #);

    #);
  toggleitem: item
  (* Special item that is checked evey second time it is selected *)
    (# onSelect::<  (#  ... #)
    #);
  newseparator:
  (* Open and append a separator to THIS(simplemenu) *)
    (#  ... #);
  new:
  (* Open THIS(simplemenu) and append it to THIS(MenuBar)
   * with label menutext.
   *)
    (# menutext: ^text;
    enter menutext[]
    ...
    #);
  changechecked:
  (* Explicitly change the checked item to be newitem *)
    (# newitem: ^radioitem
    enter newitem[]
    ...
    #);
  <<SLOT SimpleMenuLib:Attributes>>;
  private: @...;

#)
```

21.27 Streamwindow Interface

```
ORIGIN '~beta/guienv/guienv';
LIB_DEF 'guienvutilsstreamwin' '../../lib';
BODY 'private/streamwindowbody';

--GuienvLib: attributes--
streamwindow: window
  (* A separate scrolled window, which displays the text put to the
   * stream 'theStream'.
   * theStream buffers output to prevent too frequent
   * updates of the text output.
   * Use flush to ensure everything is displayed in
   * text output.
   * Should only be 'open'-ed once. Call 'show' and 'hide'
   * to make the window visible/invisible.
   *)
  (# theStream: @stream<<SLOT StreamWindowStreamBody: MainPart>>;
     close::<(# ... #);
     eventHandler::<
       (#
          onAboutToClose::< (# ... #);
       #);
     open::< (# ... #);
     show: ...;
     flush: ...;
     vanish: (* deletes logfile, if opened *)
       (# ... #);
     logfilename: ^text
       (* If specified, logs output to file with that name too *);
     hideOnCloseByUser:@Boolean;
       (* If true then window will be hide when a user "click" on close *);
     saveAs:
       (* Opens a standard file dialog which prompts for a filename.
        * If this dialog is confirmed, the text written to THIS(StreamWindow)
        * is written to the named file.
        *)
       (# dialogtitle: ^text;
       enter dialogtitle[]
       do ...
       #);
     private: @...;
  exit theStream[]
  #)
```

21.28 TabControl Interface

```
ORIGIN '~beta/guienv/guienv';
LIB_DEF 'guienvutilstabctrl' '../../lib';
BODY 'private/tabControlBody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1997-98
 *       All rights reserved.
 *)

-- windowLib: attributes --

tabControl: canvas
  (# <<SLOT tabControlLib: attributes>>;

     resourceAllocationError:< exception
       (# errorMsg: ^text;
       enter errorMsg[]
       do INNER;
       #);

     apiError:< exception
       (# errorMsg: ^text;
       enter errorMsg[]
       do INNER;
       #);

     useError:< exception
       (# errorMsg: ^text;
       enter errorMsg[]
       do INNER;
       #);

     multiLine:<
       (* if multiLine is TRUE the tabControl displays multiple
        * rows of tabs, if necessary, so all tabs are visible at once.
        *)
       booleanValue;

     tab:
       (# label:
            (# theLabel: ^text;
            enter (# enter theLabel[] ... #)
            exit (# ... exit theLabel[] #)
            #);
          icon:
            (# theIcon: ^pixmap;
            enter (# enter theIcon[] ... #)
            exit (# ... exit theIcon[] #)
            #);
          page:
            (# thePage: ^canvas;
            enter (# enter thePage[] ... #)
            exit (# ... exit thePage[] #)
            #);

          open:<(# ... #);

          close:<
            (# ... #);

          select:
            (# ... #);
```

```
      onSelect:<
        (* Called when the selection has been changed by the end user *)
        (#
        do INNER onSelect;
        #);

      tabPrivate: @...;

  #);

tabs: @
  (#
     size: integerValue
       (# ... #);

     scan:
       (# current: ^tab;
       ...
       #);

     tabsPrivate: @...;
  #);

open::<
  (# create::<
       (# ... #);
  ...
  #);

close::<
  (# ... #);

onBeforeSelectionChange:<
  (* Called when the selection is about to change;
   * Assign FALSE to allow to prevent the selection from changing.
   *)
  (# allow: @boolean;
  do TRUE -> allow;
     INNER;
  exit allow
  #);

selection:
  (# theTab: ^tab;
  enter (# enter theTab[] ... #)
  exit (# ... exit theTab[] #)
  #);

eventHandler::<
  (# onMouseUp::<
       (# ... #);
     onMouseDown::<
       (# ... #);
     onFatherFrameChanged::<
       (# ... #);
     onActivate::<
       (# ... #);
            onRefresh::<
                (# ... #);
  #);
private: @...;
#)
```

21.29 Tableview Interface

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/graphics';
BODY 'private/tableviewbody';
-- windowLib: Attributes --
tableView: canvas
  (#
     <<SLOT tableViewLib:Attributes>>;
     tableItemType:< tableItem;
     abstractColumn: interfaceObject
       (#
          <<SLOT abstractColumnLib:Attributes>>;
          title:
            (# value: ^text;
            enter
              (#  enter value[] ... #)
            exit
              (#
              ...
              exit value[]
              #)
            #);
          width:
            (# value: @integer;
            enter (#  enter value ... #)
            exit (#  ... exit value #)
            #);
          less:< booleanValue
            (#
               left,right: ^tableItemType;
               leftData,rightData: ^text;
               doneInInner: @boolean;

            enter (left[],right[])
            ...
            #);
          open::< (#
            ...
            #);
          close::<  (#  ... #);
          eventHandler::<
            (#
               drawTitleEvent: event
                 (#
                    g: ^graphics;
                    bounds: ^rectangle;

                 enter (g[],bounds[])
                 do INNER ;
                 #);
               onDrawTitle:< drawTitleEvent;
               drawItemEvent: event
                 (#
                    g: ^graphics;
                    bounds: ^rectangle;
                    theTableItem:
                      ^tableItemType;
                    data: ^text;

                 enter (g[],bounds[],theTableItem[],data[])
                 do INNER ;
                 #);
               onDrawItem:< drawItemEvent;
```

```
        #);
      private: @...;

  #);
appendColumn:
  (#
     theColumn: ^abstractColumn;
  enter theColumn[]
  ...
  #);
insertColumn:
  (# theColumn: ^abstractColumn; after: ^abstractColumn;
  enter (theColumn[],after[])
  ...
  #);
deleteColumn:
  (# theColumn: ^abstractColumn;
  enter theColumn[]
  ...
  #);
horizontalScrollbarVisible:
  (# visible: @boolean;
  enter
    (#
    enter visible
    ...
    #)
  exit
    (#
    ...
    exit visible
    #)
  #);
verticalScrollbarVisible:
  (# visible: @boolean;
  enter
    (#
    enter visible
    ...
    #)
  exit
    (#
    ...
    #)
  #);
changeHiliteColor:
  (# aColor: @Color
  enter aColor
  ...
  #);
columnWithIconType:<
 abstractColumn
  (#
     eventHandler::<
       (#
          onDrawTitle::<
            (#  ... #);
          onDrawItem::<
            (#
            ...
            #);

       #);

  #);
```

```
columnWithIcon: @columnWithIconType;
alignLeft: (#  exit 1 #);
alignRight: (#  exit 2 #);
column: abstractColumn
  (#
     <<SLOT columnnLib:Attributes>>;
     alignment:
       (# value: @integer;
       enter
         (#
         enter value
         ...
         #)
       exit (#  ... exit value #)
       #);
     open::<
       (#
       ...
       #);
     close::<
       (#
       ...
       #);
     eventHandler::<
       (#
          onDrawTitle::<  (#  ... #);
          onDrawItem::<
            (#
            ...
            #);

       #);
     private: @...;

  #);
tableItem: interfaceObject
  (#
     <<SLOT tableItemLib:Attributes>>;
     mainText:
       (# value: ^text;
       enter (#  enter value[] ... #)
       exit (#  ... exit value[] #)
       #);
     icon:
       (# theIcon: ^pixmap;
       enter (#  enter theIcon[] ... #)
       exit
         (#
         ...
         exit theIcon[]
         #)
       #);
     setColumnText:
       (# theColumn: ^abstractColumn; value: ^text;
       enter (theColumn[],value[])
       ...
       #);
     getColumnText:
       (#
          theColumn:
            ^abstractColumn;
          value: ^text;

       enter theColumn[]
       ...
       exit value[]
```

```
        #);
      textIndent:
        (# value: @integer;
        enter (#  enter value ... #)
        exit (#  ... exit value #)
        #);
      iconIndent:
        (# value: @integer;
        enter (#  enter value ... #)
        exit (#  ... exit value #)
        #);
      selected: booleanValue (#  ... #);
      eventHandler::<
        (#
           selectEvent: event
             (#  do INNER ;  #);
           onSelect:< selectEvent
             (# doubleClick: @boolean;
             enter doubleClick
             do INNER
             #);
           deSelectEvent: event
             (# do INNER #);
           onDeSelect:< deSelectEvent
        #);
      open::<
        (#
        ...
        #);
      close::<  (#  ... #);
      private:
        @...;

  #);
append:
  (# theTableItem: ^tableItemType;
  enter theTableItem[]
  ...
  #);
prepend:
  (# theTableItem: ^tableItemType;
  enter theTableItem[]
  ...
  #);
delete:
  (# theTableItem: ^tableItemType;
  enter theTableItem[]
  ...
  #);
insert:
  (#
     theTableItem: ^tableItemType;
     after: ^tableItemType;

  enter (theTableItem[],after[])
  ...
  #);
scan: (# current: ^tableItemType;  ... #);
deleteSelection:
  (#
  ...
  #);
selection: @
  (#
     set:
       (# theTableItem: ^tableItemType;
```

```
      enter theTableItem[]
      ...
      #);
    add:
      (#
         theTableItem:
           ^tableItemType;

      enter theTableItem[]
      ...
      #);
    remove:
      (#
         theTableItem:
           ^tableItemType;

      enter theTableItem[]
      ...
      #);
    clear:
      (#
      ...
      #);
    scan:
      (# current: ^tableItemType;
      ...
      #);

  #);
theScroll:
  (# x,y: @integer;
  enter
    (#
    enter (x,y)
    ...
    #)
  exit (#  ... exit (x,y) #)
  #);
scroll:
  (# dX,dY: @integer;
  enter (dX,dY)
  ...
  #);
maxScroll:
  (# maxX,maxY: @integer;
  ...
  exit (maxX,maxY)
  #);
sortByColumn:
  (# theColumn: ^column;
  enter
    (#
    enter theColumn[]
    ...
    #)
  exit (#  ... exit theColumn[] #)
  #);
open::<  (#  ... #);
close::<
  (#  ... #);
eventHandler::<

  (#
     onFrameChanged::<  (#  ... #);
     onMouseUp::<  (#  ... #);
```

```
      #);
    private:
      @...;

#)
```

21.30 Textfieldadds Interface

```
ORIGIN '../fields';
LIB_DEF 'guienvutilstextfld' '../../lib';
MDBODY default 'private/X11/textfieldadds_X11body'
       mac 'private/macintosh/textfieldadds_macbody'
       ppc 'private/macintosh/textfieldadds_macbody'
       ppcmac 'private/macintosh/textfieldadds_macbody'
       nti 'private/winnt/textfieldadds_ntibody';
-- textFieldLib: Attributes --
posToRowCol:
  (#
     pos: @integer;
     row,col: @integer;
     indexError:< exception
       (#  do 'pos out of range in posToRowCol'->msg.append; INNER ;  #);

  enter pos
  do ...;
  exit (row,col)
  #);
rowColToPos:
  (#
     row,col: @integer;
     pos: @integer;
     indexError:< exception
       (#
       do
          'Row or col out of range error in rowColToPos'->msg.append;
          INNER ;

       #);

  enter (row,col)
  do ...;
  exit pos
  #);
automaticScrolling:
  (# value: @boolean;
  enter value
  ...
  #);
disableUpdate: (#  ... #);
enableUpdate:
  (#
  ...
  #);
updateLine:
  (* updateLine updates lineNumber in THIS(textField)
   *)
  (# lineNumber: @integer
  enter lineNumber
  ...
  #);
updateRegion:
  (* updateRegion updates the textfield from
   * (0,y1) to (size.x,y2)
   * NB: the coordinates is in pixels
   *)
  (# y1,y2: @integer
  enter (y1,y2)
  ...
  #)
```

21.31 Tooltip Interface

```
ORIGIN '~beta/guienv/guienv';
BODY 'private/tooltipBody';

INCLUDE '~beta/guienv/graphmath';

-- windowlib: attributes --
tool:
  (# theWindowItem: ^windowItem;
     theTip: ^text;
  #);

rectTool: tool
  (#
     theRect: ^rectangle; (* Coordinates of the bounding rectangle of the
                           * tool.
                           * The coordinates are relative to the upper-left
                           * corner theWindowItem.
                           * NOT supported on unix.
                           *)
  #);

toolTipControl:
  (#
     <<SLOT GUIENVtooltipLib: attributes>>;


     (* Duration types  used in setDelayTime. *)

     AUTOMATIC: (# exit 0 #); (* Automatically calculates the initial,
                               * reshow, and autopopup durations based
                               * on the value of delay. *)

     RESHOW: (# exit 1 #);  (* Sets the length of the delay before
                             * subsequent ToolTip windows are displayed
                             * when the cursor is moved from one tool
                             * to another.*)

     AUTOPOP: (# exit 2 #);  (* Sets the length of time before the
                              * ToolTip window is hidden if the cursor
                              * remains stationary in the tool's
                              * bounding rectangle after the ToolTip
                              * window has appeared. *)

     INITIAL: (# exit 3 #);  (* Sets the length of time that the
                              * cursor must remain stationary within
                              * the bounding rectangle of a tool before
                              * the ToolTip window is displayed. *)

     setDelayTime:
       (# delay: @integer; (* New duration, in milliseconds. *)
          durationType: @integer;
       enter (durationType, delay)
       ...
       #);


     add:
       (# theTool: ^tool;
       enter theTool[]
       ...
       #);
```

```
      delete:
        (# theTool: ^tool;
        enter theTool[]
        ...
        #);

      open:<(# ... #);

      close:<
        (# ... #);

      tooltipPriv: @...;
  #);

-- windowitemlib:attributes--
addToolTip:
  (# t:^text;
     thetool:^tool;
     tooltipc:^tooltipcontrol;
  enter t[]
     ...
  #);
addToolTipExt:
  (# t:^text;
     thetool:^tool;
     theWindow:^window;
     tooltipc:^tooltipcontrol;
  enter (theWindow[], t[])
     ...
  #)
```

21.32 Treeview Interface

```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/fields';

BODY 'private/treeviewbody';

--- windowlib: attributes ---
treeview: canvas
  (# <<SLOT treeviewLib: attributes>>;
     selection: ^item;
     item: (* abstract superpattern for elements in the tree *)
       (# <<SLOT treeviewItemLib: attributes>>;
          isFolder:< booleanValue;
          father: ^folder (* the folder in which this(item) is
                           * located - this attribute is only valid,
                           * if this(item) is added to the folder
                           * using addItem
                           *);
          icon:< (* the icon to show for this type of item *)
            (# i: ^guienv.pixmap
            enter i[]
            ...
            #);
          label: ^text (* the text of this item *);
          popupmenu:< (* popup menu on this item *)
            (# m: ^guienv.menu
            enter m[]
            ...
            exit m[]
            #);
          scrollIntoView:
            (# sc: ^scroller
            enter sc[]
            ...
            #);
          select:
            (# ... #);
          deselect:
            (# ... #);
          onIconSelected:< (* invoked when item icon is selected *)
            (# ev: ^eventhandler.mouseevent
            enter ev[]
            ...
            #);
          onSelected:< (* invoked when item is selected *)
            (# ev: ^eventhandler.mouseevent
            enter ev[]
            ...
            #);
          onLabelChanged:<
            (* invoked the label have been edited.  newLabel will
             * contain the edited label value.
             *)
            (# newLabel: ^text
            enter newLabel[]
            ...
            #);
          allowLabelEdit:< booleanValue;
          itemPrivate: @...;
       #);
     folder: item
       (# <<SLOT treeviewFolderLib: attributes>>;
          isFolder:: trueObject;
```

```
      items: @list(# element:: item #);
      addItem:
        (# i: ^item
        enter i[]
        do this(folder)[]->i.father[];
           i[]->items.append;
           INNER
        #);
      prependItem:
        (# i: ^item
        enter i[]
        do this(folder)[]->i.father[];
           i[]->items.prepend;
           INNER
        #);
      Icon::<(* the icon to show for this folder when open *)
        (# openicon: ^guienv.pixmap
        enter openicon[]
        ...
        #);
      onToggleSelected:< (* invoked when toggle icon is selected *)
        (# ... #);
      onIconSelected::<(# ... #);
      onSelected::<
        (# ... #);
      onExpand:< (* invoked when folder is opened *)
        (# ... #);
      onCollapse:< (* invoked when folder is closed again *)
        (# ... #);
      isExpanded:<
        (# open:@boolean
          ...
        exit open
        #);
      folderPrivate: @...;
  #);
root:
  (# f: ^folder
  enter (# enter f[] ... #)
  exit (# ... exit f[] #)
  #);
hideRoot:< (* if trueObject then root is not shown *)
  booleanValue;
minSize:< (* Specifies the minimum size of this(treeView) *)
  (# width, height: @integer;
  do INNER;
  exit (width, height)
  #);
changed:
  (# ... #);
eventHandler::<
  (# onRefresh::<
        (# ... #);
      onMouseDown::<
        (# ... #);
      onMouseUp::<
        (# ... #);
      onUpdate:< (* further bind onUpdate and change etc. newheight,
                  * to change the size of this treeview
                  *)
        (# width,height:@integer;
           newwidth,newheight:@integer;
           updateDoneInInner:@boolean;
        enter (width,height)
        do width->newwidth;
           height->newheight;
```

```
            INNER;
            (if not updateDoneInInner then
                (if (newwidth<>width) or (newheight<>height) then
                    (newwidth,newheight)->size
                 else
                    true->update;
                if);
            if)
         #)
     #);
   open::<
     (# ... #);
   treeviewPrivate: @...;
#)
```

21.33 Walkingants Interface

```
ORIGIN '../guienv';
LIB_DEF 'guienvutilsants' '../../lib';
BODY 'private/walkingantsbody';

-- GUIenvLib: attributes --

oneWay: (# exit 1 #);
twoWay: (# exit 2 #);

walkingAnts:
  (* WalkingsAnts provides a way to highlight an area inside a windowItem.
   *
   * The walking ants are an animated stippled rectangle.
   *
   * Interval is the pause in microseconds between each step.
   *
   * More than one area can be animated at once.
   *
   *)
  (# <<SLOT walkingAntsLib: attributes>>;

     (* Interval is the pause in microseconds
      * between each step.
      *)
     interval: @integer;

     (* Style is oneWay or twoWay.
      *   oneWay: the ants walks clockwise around the edge
      *           of the area.
      *   twoWay: the ants alternates between clock
      *           and counterclock wise.
      *)
     style: @integer;

     (* Size (width and hight) of pen to use *)
     pensize: @integer;

     init:<(#
       ...
       #);

     add:
       (* Add an area. *)
       (# theRectangle: ^rectangle;
          theWindowItem: ^window.windowItem;
       enter (theWindowItem[], theRectangle[])
       ...
       #);
     delete:
       (* Delete an area. *)
       (# theRectangle: ^rectangle;
          theWindowItem: ^window.windowItem;
       enter (theWindowItem[], theRectangle[])
       ...
       #);
     clear:
       (* Delete all areas *)
       (# ... #);

     start:
       (* Start the animation *)
       (#
```

```
      ...
      #);
    stop:
      (* Stop the animation *)
      (# ... #);

    private: @...;
#)
```

22.1 Cdplayer Interface

```
ORIGIN '~beta/guienv/guienv';
BODY 'private/cdplayerbody';

-- guienvLib: attributes --


cdPlayer:
  (* This is an interface to controlling the CD player
   * build into the computer.
   *)
  (# <<SLOT cdPlayerLib: attributes>>;
     deviceError:< exception
       (# errMsg: ^text;
       enter errMsg[]
       do INNER deviceError
       #);
     allocationError:< exception
       (# errMsg: ^text;
       enter errMsg[]
       do INNER allocationError
       #);
     open:<(# ... #);
     close:<
       (# ... #);
     cdPresent:
       (* Tells if a cd is present in the cd drawer. *)
       (# isCDPresent: @boolean;
       ...
       exit isCDPresent
       #);
     play:
       (* Play starting at the specified track number. *)
       (# track: @integer;
       enter track
       ...
       #);
     stop:
       (* Stop playing, rewinding to the start of the CD *)
       (#
       ...
       #);
     pause:
       (* Pause playing. *)
       (#
       ...
       #);
     resume:
       (* Resume playing. *)
       (#
       ...
       #);
     duration:
       (* Return the total running time of the current CD in milliseonds. *)
       (# theDuration: @integer;
       ...
       exit theDuration
       #);
     currentTime:
       (* This is the current time in milliseconds. *)
       (# theTime: @integer;
          theTrack:@integer;
       enter (# enter (theTrack,theTime) ... #)
```

```
      exit (# ... exit (theTrack,theTime) #)
      #);
    numberOfTracks:
      (* Return the number of tracks on the current CD. *)
      (# tracks: @integer;
      ...
      exit tracks
      #);
    private: @...;
#)
```

22.2 Movieplayer Interface

```
ORIGIN '~beta/guienv/guienv';

BODY 'private/movieplayerbody';


-- guienvLib: attributes --

movie:
  (* A movie represents quicktime movies that can be
   * played in a moviePlayer.
   *)
  (# <<SLOT movieLib: attributes>>;
     read:
       (* Read the specified movie file. *)
       (# theFileName: ^text;
       enter theFileName[]
       ...
       #);
     userSelectFile:
       (* Let the user select a movie file in a standard file
        * dialog.
        *)
       (# success: @boolean;
       ...
       exit success
       #);
     duration:
       (* Return the duration of the movie in milliseconds *)
       (# theDuration: @integer;
       ...
       exit theDuration
       #);
     fileName:
       (* The name of the movie file. *)
       (# theFileName: ^text;
       ...
       exit theFileName[]
       #);
     displaySize:
       (* The natural width and height of the movie. *)
       (# width, height: @integer;
       ...
       exit (width, height)
       #);
     private: @...;
  #);


-- windowLib: attributes --

moviePlayer: windowitem
  (# <<SLOT moviePlayerLib: attributes>>;
     controllerVisible:< (# value: @boolean; do TRUE -> value; INNER exit value#);
                 (* Furtherbind  controllerVisible to "false -> value"
                  * to hide the controller.
                     *)
     automaticResize:< (# value: @boolean; do TRUE -> value; INNER exit value#);
                 (* Furtherbind  automaticResize to "false -> value"
                  * to prevent the moviePlayer from resizing autmatically
                     * based on the natural size of the movie.
                     *)
     contents:
```

```
  (* The movie that are selected in the moviePlayer. *)
  (#
  enter (# enter  theMovie[] ... #)
  exit theMovie[]
  #);
read:
  (* Read the specified movie file. This is a shortcut for initialising a
   * Movie.
   *)
  (# theFileName: ^text;
  enter theFileName[]
  ...
  #);
userSelectFile:
  (* Let the user select a movie file in a standard file
   * dialog.This is a shortcut for initialising a
   * Movie.
   *)
  (# success: @boolean;
  ...
  exit success
  #);

currentTime:
  (* The current time  is measured in milliseconds *)
  (# theTime: @integer;
  enter (# enter theTime ... #)
  exit (# ... exit theTime #)
  #);
currentFrame:
  (* The current frame *)
  (# theFrame: @integer;
  enter (# enter theFrame ... #)
  exit (# ... exit theFrame #)
  #);

play:
  (* Start playing. *)
  (# ... #);
stop:
  (* Stop playing. *)
  (# ... #);
pause:
  (* Pause playing. *)
  (# ... #);
open::<
  (# create::(# ...#);
  ...
  #);
close::<
  (#
  ...
  #);
eventHandler::<
  (# contentsChanged: event
       (# name:^text;
       enter name[]
       do INNER;
       #);
     onContentsChanged:< contentsChanged;

     onActivate::<
       (# ... #);
     onDeactivate::<
       (# ... #);
     onMouseDown::<
```

```
          (# ... #);
        onRefresh::<
          (# ... #);
      #);
   theMovie: ^movie;
   private: @...;
#)
```

23.1 Opengl Interface

```
 ^movie;
     private: ^movie;
     private:private:external/betaopengl.c' 'mwcppc -w off -o $0 $1'
      sgi '$$/betaopengl.o' 'private/external/betaopengl.c' '$CC -32 -D$$ -c -o $0 $1'
      macosx '$$/betaopengl.o' 'private/external/betaopengl.c' '$CC -c -D$$ -o $0 $1'
      sun4s '$$/betaopengl.o' 'private/external/betaopengl.c' '$CC -c -D$$ -o $0 $1'
      linux '$$/betaopengl.o' 'private/external/betaopengl.c' '$CC -c -D$$ -o $0 $1';



LINKOPT macosx '-framework opengl -framework GLUT -framework Cocoa'
        nti_ms  'OPENGL32.lib GLU32.lib'
        nti_gnu '-lopengl32 -lglu32'
        sun4s '-lGLw -lGLU -lGL'
        linux '-lGL -lGLU -lglut -lX11 -lXm -lXext -lXi'
        sgi '-lGLw -lGLU -lGL -lXm -lXt -lX11 -lXext'
        default '';


--LIB: attributes--

(* Disclaimer:
 * This is a prelease version of the openGL
 * library interface for BETA. It may be
 * changed later, and there is some inconveniences.
 *
 * Fell free to ask questions and come with proposals to
 * support@mjolner.dk
 *
 * OpenGL operates with a large number of types, these
 * are mapped to beta types as follows:
 * |-------------------------------------------------------|
 * |c-type        | openGL       | Beta        | size in bytes|
 * |-------------------------------------------------------|
 * |void          | GLvoid       |             |              |
 * |unsigned char | GLboolean    | int8u       |  1           |
 * |signed char   | GLbyte       | int8        |  1           |
 * |short         | GLshort      | int16       |  2           |
 * |int           | GLint        | int32       |  4           |
 * |unsigned char | GLubyte      | int8u       |  1           |
 * |unsigned char*| GLubyte*     | [0]@char    |  1           |
 * |unsigned short| GLushort     | int16u      |  2           |
 * |unsigned int  | GLuint       | int32u      |  4           |
 * |         int  | GLsizei      | int32       |  4           |
 * |float         | GLfloat      | real        |  8           |
 * |float         | GLclampf     | real        |  8           |
 * |double        | GLdouble     | real        |  8           |
 * |double        | GLclampd     | real        |  8           |
 * |unsigned int  | GLbitfield   | int32u      |  4           |
 * |unsigned int  | GLenum       | int32u      |  4           |
 * |xxx*          | xxx*         | int32       |  4           |
 * |xxx**         | xxx**        | int32       |  4           |
 * -------------------------------------------------------|
 *
 * Original GL-types are shown as comments in patterns.
 *
 * Notice that GLfloat's are mapped to reals. These reals
 * are converted to floats in a c program, just before
 * actually calling the openGL routines. This is of
 * course expensive. Use the GLdouble routines when
 * they are available.
 *
```

```
 * Also functions that float arrays are expensive
 * because the BETA repetion of reals is copied and
 * and converted to a c−array of floats. In this prelease
 * this is only done when absolutely necessary. That is
 * glVertex3fv is not working, as you can use glVertex3f.
 * But glLightf does converting. Allways use repetitions,
 * that is as long as the longest parameter ever. (4 in glLightfv)
 *
 * Functions that return arrays are generally not
 * working. This will be better later.
 *
 * Some funtions are defined equally. eg. glRotatef is the
 * same pattern as glRotated. There is no need for a
 * glRotatef as you can not have any floats in your
 * beta program. So they are defined:
 * glRotated:(# ... #);
 * glRotatef:glRotated(# #);
 * This is done only to ease porting of c−programs.
 *
 * logical or '|' is not part of the "highlevel" BETA syntax.
 * eg.  c:
 * glEnable(GL_LIGHTING|GL_CULLFACE);
 * in BETA:
 * GL_LIGHTING−>glEnable;
 * GL_CULLFACE−>glEnable;
 * or
 * GL_LIGHTING %Bor GL_CULLFACE −> glEnable;
 *)

(*
 *
 * Enumerations
 *
 *)

GL_FALSE:                       (# exit 0 #);
GL_TRUE:                        (# exit 1 #);

(* Data types *)
GL_BYTE:                        (# exit 0x1400 #);
GL_UNSIGNED_BYTE:               (# exit 0x1401 #);
GL_SHORT:                       (# exit 0x1402 #);
GL_UNSIGNED_SHORT:              (# exit 0x1403 #);
GL_INT:                         (# exit 0x1404 #);
GL_UNSIGNED_INT:                (# exit 0x1405 #);
GL_FLOAT:                       (# exit 0x1406 #);
GL_DOUBLE:                      (# exit 0x140A #);
GL_2_BYTES:                     (# exit 0x1407 #);
GL_3_BYTES:                     (# exit 0x1408 #);
GL_4_BYTES:                     (# exit 0x1409 #);

(* Primitives *)
GL_LINES:                       (# exit 0x0001 #);
GL_POINTS:                      (# exit 0x0000 #);
GL_LINE_STRIP:                  (# exit 0x0003 #);
GL_LINE_LOOP:                   (# exit 0x0002 #);
GL_TRIANGLES:                   (# exit 0x0004 #);
GL_TRIANGLE_STRIP:              (# exit 0x0005 #);
GL_TRIANGLE_FAN:                (# exit 0x0006 #);
GL_QUADS:                       (# exit 0x0007 #);
GL_QUAD_STRIP:                  (# exit 0x0008 #);
GL_POLYGON:                     (# exit 0x0009 #);
GL_EDGE_FLAG:                   (# exit 0x0B43 #);

(* Vertex Arrays *)
GL_VERTEX_ARRAY:                (# exit 0x8074 #);
```

```
GL_NORMAL_ARRAY:                (# exit 0x8075 #);
GL_COLOR_ARRAY:                 (# exit 0x8076 #);
GL_INDEX_ARRAY:                 (# exit 0x8077 #);
GL_TEXTURE_COORD_ARRAY:         (# exit 0x8078 #);
GL_EDGE_FLAG_ARRAY:             (# exit 0x8079 #);
GL_VERTEX_ARRAY_SIZE:           (# exit 0x807A #);
GL_VERTEX_ARRAY_TYPE:           (# exit 0x807B #);
GL_VERTEX_ARRAY_STRIDE:         (# exit 0x807C #);
GL_NORMAL_ARRAY_TYPE:           (# exit 0x807E #);
GL_NORMAL_ARRAY_STRIDE:         (# exit 0x807F #);
GL_COLOR_ARRAY_SIZE:            (# exit 0x8081 #);
GL_COLOR_ARRAY_TYPE:            (# exit 0x8082 #);
GL_COLOR_ARRAY_STRIDE:          (# exit 0x8083 #);
GL_INDEX_ARRAY_TYPE:            (# exit 0x8085 #);
GL_INDEX_ARRAY_STRIDE:          (# exit 0x8086 #);
GL_TEXTURE_COORD_ARRAY_SIZE:    (# exit 0x8088 #);
GL_TEXTURE_COORD_ARRAY_TYPE:    (# exit 0x8089 #);
GL_TEXTURE_COORD_ARRAY_STRIDE:  (# exit 0x808A #);
GL_EDGE_FLAG_ARRAY_STRIDE:      (# exit 0x808C #);
GL_VERTEX_ARRAY_POINTER:        (# exit 0x808E #);
GL_NORMAL_ARRAY_POINTER:        (# exit 0x808F #);
GL_COLOR_ARRAY_POINTER:         (# exit 0x8090 #);
GL_INDEX_ARRAY_POINTER:         (# exit 0x8091 #);
GL_TEXTURE_COORD_ARRAY_POINTER: (# exit 0x8092 #);
GL_EDGE_FLAG_ARRAY_POINTER:     (# exit 0x8093 #);
GL_V2F:                         (# exit 0x2A20 #);
GL_V3F:                         (# exit 0x2A21 #);
GL_C4UB_V2F:                    (# exit 0x2A22 #);
GL_C4UB_V3F:                    (# exit 0x2A23 #);
GL_C3F_V3F:                     (# exit 0x2A24 #);
GL_N3F_V3F:                     (# exit 0x2A25 #);
GL_C4F_N3F_V3F:                 (# exit 0x2A26 #);
GL_T2F_V3F:                     (# exit 0x2A27 #);
GL_T4F_V4F:                     (# exit 0x2A28 #);
GL_T2F_C4UB_V3F:                (# exit 0x2A29 #);
GL_T2F_C3F_V3F:                 (# exit 0x2A2A #);
GL_T2F_N3F_V3F:                 (# exit 0x2A2B #);
GL_T2F_C4F_N3F_V3F:             (# exit 0x2A2C #);
GL_T4F_C4F_N3F_V4F:             (# exit 0x2A2D #);

(* Matrix Mode *)
GL_MATRIX_MODE:                 (# exit 0x0BA0 #);
GL_MODELVIEW:                   (# exit 0x1700 #);
GL_PROJECTION:                  (# exit 0x1701 #);
GL_TEXTURE:                     (# exit 0x1702 #);

(* Points *)
GL_POINT_SMOOTH:                (# exit 0x0B10 #);
GL_POINT_SIZE:                  (# exit 0x0B11 #);
GL_POINT_SIZE_GRANULARITY:      (# exit 0x0B13 #);
GL_POINT_SIZE_RANGE:            (# exit 0x0B12 #);

(* Lines *)
GL_LINE_SMOOTH:                 (# exit 0x0B20 #);
GL_LINE_STIPPLE:                (# exit 0x0B24 #);
GL_LINE_STIPPLE_PATTERN:        (# exit 0x0B25 #);
GL_LINE_STIPPLE_REPEAT:         (# exit 0x0B26 #);
GL_LINE_WIDTH:                  (# exit 0x0B21 #);
GL_LINE_WIDTH_GRANULARITY:      (# exit 0x0B23 #);
GL_LINE_WIDTH_RANGE:            (# exit 0x0B22 #);

(* Polygons *)
GL_POINT:                       (# exit 0x1B00 #);
GL_LINE:                        (# exit 0x1B01 #);
GL_FILL:                        (# exit 0x1B02 #);
GL_CCW:                         (# exit 0x0901 #);
```

```
GL_CW:                          (# exit 0x0900 #);
GL_FRONT:                       (# exit 0x0404 #);
GL_BACK:                        (# exit 0x0405 #);
GL_CULL_FACE:                   (# exit 0x0B44 #);
GL_CULL_FACE_MODE:              (# exit 0x0B45 #);
GL_POLYGON_SMOOTH:              (# exit 0x0B41 #);
GL_POLYGON_STIPPLE:             (# exit 0x0B42 #);
GL_FRONT_FACE:                  (# exit 0x0B46 #);
GL_POLYGON_MODE:                (# exit 0x0B40 #);
GL_POLYGON_OFFSET_FACTOR:       (# exit 0x8038 #);
GL_POLYGON_OFFSET_UNITS:        (# exit 0x2A00 #);
GL_POLYGON_OFFSET_POINT:        (# exit 0x2A01 #);
GL_POLYGON_OFFSET_LINE:         (# exit 0x2A02 #);
GL_POLYGON_OFFSET_FILL:         (# exit 0x8037 #);

(* Display Lists *)
GL_COMPILE:                     (# exit 0x1300 #);
GL_COMPILE_AND_EXECUTE:         (# exit 0x1301 #);
GL_LIST_BASE:                   (# exit 0x0B32 #);
GL_LIST_INDEX:                  (# exit 0x0B33 #);
GL_LIST_MODE:                   (# exit 0x0B30 #);

(* Depth buffer *)
GL_NEVER:                       (# exit 0x0200 #);
GL_LESS:                        (# exit 0x0201 #);
GL_GEQUAL:                      (# exit 0x0206 #);
GL_LEQUAL:                      (# exit 0x0203 #);
GL_GREATER:                     (# exit 0x0204 #);
GL_NOTEQUAL:                    (# exit 0x0205 #);
GL_EQUAL:                       (# exit 0x0202 #);
GL_ALWAYS:                      (# exit 0x0207 #);
GL_DEPTH_TEST:                  (# exit 0x0B71 #);
GL_DEPTH_BITS:                  (# exit 0x0D56 #);
GL_DEPTH_CLEAR_VALUE:           (# exit 0x0B73 #);
GL_DEPTH_FUNC:                  (# exit 0x0B74 #);
GL_DEPTH_RANGE:                 (# exit 0x0B70 #);
GL_DEPTH_WRITEMASK:             (# exit 0x0B72 #);
GL_DEPTH_COMPONENT:             (# exit 0x1902 #);

(* Lighting *)
GL_LIGHTING:                    (# exit 0x0B50 #);
GL_LIGHT0:                      (# exit 0x4000 #);
GL_LIGHT1:                      (# exit 0x4001 #);
GL_LIGHT2:                      (# exit 0x4002 #);
GL_LIGHT3:                      (# exit 0x4003 #);
GL_LIGHT4:                      (# exit 0x4004 #);
GL_LIGHT5:                      (# exit 0x4005 #);
GL_LIGHT6:                      (# exit 0x4006 #);
GL_LIGHT7:                      (# exit 0x4007 #);
GL_SPOT_EXPONENT:               (# exit 0x1205 #);
GL_SPOT_CUTOFF:                 (# exit 0x1206 #);
GL_CONSTANT_ATTENUATION:        (# exit 0x1207 #);
GL_LINEAR_ATTENUATION:          (# exit 0x1208 #);
GL_QUADRATIC_ATTENUATION:       (# exit 0x1209 #);
GL_AMBIENT:                     (# exit 0x1200 #);
GL_DIFFUSE:                     (# exit 0x1201 #);
GL_SPECULAR:                    (# exit 0x1202 #);
GL_SHININESS:                   (# exit 0x1601 #);
GL_EMISSION:                    (# exit 0x1600 #);
GL_POSITION:                    (# exit 0x1203 #);
GL_SPOT_DIRECTION:              (# exit 0x1204 #);
GL_AMBIENT_AND_DIFFUSE:         (# exit 0x1602 #);
GL_COLOR_INDEXES:               (# exit 0x1603 #);
GL_LIGHT_MODEL_TWO_SIDE:        (# exit 0x0B52 #);
GL_LIGHT_MODEL_LOCAL_VIEWER:    (# exit 0x0B51 #);
GL_LIGHT_MODEL_AMBIENT:         (# exit 0x0B53 #);
```

```
GL_FRONT_AND_BACK:              (# exit 0x0408 #);
GL_SHADE_MODEL:                 (# exit 0x0B54 #);
GL_FLAT:                        (# exit 0x1D00 #);
GL_SMOOTH:                      (# exit 0x1D01 #);
GL_COLOR_MATERIAL:              (# exit 0x0B57 #);
GL_COLOR_MATERIAL_FACE:         (# exit 0x0B55 #);
GL_COLOR_MATERIAL_PARAMETER:    (# exit 0x0B56 #);
GL_NORMALIZE:                   (# exit 0x0BA1 #);

(* User clipping planes *)
GL_CLIP_PLANE0:                 (# exit 0x3000 #);
GL_CLIP_PLANE1:                 (# exit 0x3001 #);
GL_CLIP_PLANE2:                 (# exit 0x3002 #);
GL_CLIP_PLANE3:                 (# exit 0x3003 #);
GL_CLIP_PLANE4:                 (# exit 0x3004 #);
GL_CLIP_PLANE5:                 (# exit 0x3005 #);

(* Accumulation buffer *)
GL_ACCUM_RED_BITS:              (# exit 0x0D58 #);
GL_ACCUM_GREEN_BITS:            (# exit 0x0D59 #);
GL_ACCUM_BLUE_BITS:             (# exit 0x0D5A #);
GL_ACCUM_ALPHA_BITS:            (# exit 0x0D5B #);
GL_ACCUM_CLEAR_VALUE:           (# exit 0x0B80 #);
GL_ACCUM:                       (# exit 0x0100 #);
GL_ADD:                         (# exit 0x0104 #);
GL_LOAD:                        (# exit 0x0101 #);
GL_MULT:                        (# exit 0x0103 #);
GL_RETURN:                      (# exit 0x0102 #);

(* Alpha testing *)
GL_ALPHA_TEST:                  (# exit 0x0BC0 #);
GL_ALPHA_TEST_REF:              (# exit 0x0BC2 #);
GL_ALPHA_TEST_FUNC:             (# exit 0x0BC1 #);

(* Blending *)
GL_BLEND:                       (# exit 0x0BE2 #);
GL_BLEND_SRC:                   (# exit 0x0BE1 #);
GL_BLEND_DST:                   (# exit 0x0BE0 #);
GL_ZERO:                        (# exit 0 #);
GL_ONE:                         (# exit 1 #);
GL_SRC_COLOR:                   (# exit 0x0300 #);
GL_ONE_MINUS_SRC_COLOR:         (# exit 0x0301 #);
GL_DST_COLOR:                   (# exit 0x0306 #);
GL_ONE_MINUS_DST_COLOR:         (# exit 0x0307 #);
GL_SRC_ALPHA:                   (# exit 0x0302 #);
GL_ONE_MINUS_SRC_ALPHA:         (# exit 0x0303 #);
GL_DST_ALPHA:                   (# exit 0x0304 #);
GL_ONE_MINUS_DST_ALPHA:         (# exit 0x0305 #);
GL_SRC_ALPHA_SATURATE:          (# exit 0x0308 #);
GL_CONSTANT_COLOR:              (# exit 0x8001 #);
GL_ONE_MINUS_CONSTANT_COLOR:    (# exit 0x8002 #);
GL_CONSTANT_ALPHA:              (# exit 0x8003 #);
GL_ONE_MINUS_CONSTANT_ALPHA:    (# exit 0x8004 #);

(* Render Mode *)
GL_FEEDBACK:                    (# exit 0x1C01 #);
GL_RENDER:                      (# exit 0x1C00 #);
GL_SELECT:                      (# exit 0x1C02 #);

(* Feedback *)
GL_2D:                          (# exit 0x0600 #);
GL_3D:                          (# exit 0x0601 #);
GL_3D_COLOR:                    (# exit 0x0602 #);
GL_3D_COLOR_TEXTURE:            (# exit 0x0603 #);
GL_4D_COLOR_TEXTURE:            (# exit 0x0604 #);
GL_POINT_TOKEN:                 (# exit 0x0701 #);
```

```
GL_LINE_TOKEN:                  (# exit 0x0702 #);
GL_LINE_RESET_TOKEN:            (# exit 0x0707 #);
GL_POLYGON_TOKEN:               (# exit 0x0703 #);
GL_BITMAP_TOKEN:                (# exit 0x0704 #);
GL_DRAW_PIXEL_TOKEN:            (# exit 0x0705 #);
GL_COPY_PIXEL_TOKEN:            (# exit 0x0706 #);
GL_PASS_THROUGH_TOKEN:          (# exit 0x0700 #);
GL_FEEDBACK_BUFFER_POINTER:     (# exit 0x0DF0 #);
GL_FEEDBACK_BUFFER_SIZE:        (# exit 0x0DF1 #);
GL_FEEDBACK_BUFFER_TYPE:        (# exit 0x0DF2 #);


(* Fog *)
GL_FOG:                         (# exit 0x0B60 #);
GL_FOG_MODE:                    (# exit 0x0B65 #);
GL_FOG_DENSITY:                 (# exit 0x0B62 #);
GL_FOG_COLOR:                   (# exit 0x0B66 #);
GL_FOG_INDEX:                   (# exit 0x0B61 #);
GL_FOG_START:                   (# exit 0x0B63 #);
GL_FOG_END:                     (# exit 0x0B64 #);
GL_LINEAR:                      (# exit 0x2601 #);
GL_EXP:                         (# exit 0x0800 #);
GL_EXP2:                        (# exit 0x0801 #);

(* Logic Ops *)
GL_LOGIC_OP:                    (# exit 0x0BF1 #);
GL_INDEX_LOGIC_OP:              (# exit 0x0BF1 #);
GL_COLOR_LOGIC_OP:              (# exit 0x0BF2 #);
GL_LOGIC_OP_MODE:               (# exit 0x0BF0 #);
GL_CLEAR:                       (# exit 0x1500 #);
GL_SET:                         (# exit 0x150F #);
GL_COPY:                        (# exit 0x1503 #);
GL_COPY_INVERTED:               (# exit 0x150C #);
GL_NOOP:                        (# exit 0x1505 #);
GL_INVERT:                      (# exit 0x150A #);
GL_AND:                         (# exit 0x1501 #);
GL_NAND:                        (# exit 0x150E #);
GL_OR:                          (# exit 0x1507 #);
GL_NOR:                         (# exit 0x1508 #);
GL_XOR:                         (# exit 0x1506 #);
GL_EQUIV:                       (# exit 0x1509 #);
GL_AND_REVERSE:                 (# exit 0x1502 #);
GL_AND_INVERTED:                (# exit 0x1504 #);
GL_OR_REVERSE:                  (# exit 0x150B #);
GL_OR_INVERTED:                 (# exit 0x150D #);

(* Stencil *)
GL_STENCIL_TEST:                (# exit 0x0B90 #);
GL_STENCIL_WRITEMASK:           (# exit 0x0B98 #);
GL_STENCIL_BITS:                (# exit 0x0D57 #);
GL_STENCIL_FUNC:                (# exit 0x0B92 #);
GL_STENCIL_VALUE_MASK:          (# exit 0x0B93 #);
GL_STENCIL_REF:                 (# exit 0x0B97 #);
GL_STENCIL_FAIL:                (# exit 0x0B94 #);
GL_STENCIL_PASS_DEPTH_PASS:     (# exit 0x0B96 #);
GL_STENCIL_PASS_DEPTH_FAIL:     (# exit 0x0B95 #);
GL_STENCIL_CLEAR_VALUE:         (# exit 0x0B91 #);
GL_STENCIL_INDEX:               (# exit 0x1901 #);
GL_KEEP:                        (# exit 0x1E00 #);
GL_REPLACE:                     (# exit 0x1E01 #);
GL_INCR:                        (# exit 0x1E02 #);
GL_DECR:                        (# exit 0x1E03 #);

(* Buffers, Pixel Drawing/Reading *)
GL_NONE:                        (# exit 0 #);
GL_LEFT:                        (# exit 0x0406 #);
```

```
GL_RIGHT:                          (# exit 0x0407 #);

                                      (*GL_FRONT:                     (# exit 0x0404 #); *)
                                      (*GL_BACK:                      (# exit 0x0405 #); *)
                                      (*GL_FRONT_AND_BACK:            (# exit 0x0408 #); *)
GL_FRONT_LEFT:                     (# exit 0x0400 #);
GL_FRONT_RIGHT:                    (# exit 0x0401 #);
GL_BACK_LEFT:                      (# exit 0x0402 #);
GL_BACK_RIGHT:                     (# exit 0x0403 #);
GL_AUX0:                           (# exit 0x0409 #);
GL_AUX1:                           (# exit 0x040A #);
GL_AUX2:                           (# exit 0x040B #);
GL_AUX3:                           (# exit 0x040C #);
GL_COLOR_INDEX:                    (# exit 0x1900 #);
GL_RED:                            (# exit 0x1903 #);
GL_GREEN:                          (# exit 0x1904 #);
GL_BLUE:                           (# exit 0x1905 #);
GL_ALPHA:                          (# exit 0x1906 #);
GL_LUMINANCE:                      (# exit 0x1909 #);
GL_LUMINANCE_ALPHA:                (# exit 0x190A #);
GL_ALPHA_BITS:                     (# exit 0x0D55 #);
GL_RED_BITS:                       (# exit 0x0D52 #);
GL_GREEN_BITS:                     (# exit 0x0D53 #);
GL_BLUE_BITS:                      (# exit 0x0D54 #);
GL_INDEX_BITS:                     (# exit 0x0D51 #);
GL_SUBPIXEL_BITS:                  (# exit 0x0D50 #);
GL_AUX_BUFFERS:                    (# exit 0x0C00 #);
GL_READ_BUFFER:                    (# exit 0x0C02 #);
GL_DRAW_BUFFER:                    (# exit 0x0C01 #);
GL_DOUBLEBUFFER:                   (# exit 0x0C32 #);
GL_STEREO:                         (# exit 0x0C33 #);
GL_BITMAP:                         (# exit 0x1A00 #);
GL_COLOR:                          (# exit 0x1800 #);
GL_DEPTH:                          (# exit 0x1801 #);
GL_STENCIL:                        (# exit 0x1802 #);
GL_DITHER:                         (# exit 0x0BD0 #);
GL_RGB:                            (# exit 0x1907 #);
GL_RGBA:                           (# exit 0x1908 #);

(* Implementation limits *)
GL_MAX_LIST_NESTING:               (# exit 0x0B31 #);
GL_MAX_ATTRIB_STACK_DEPTH:         (# exit 0x0D35 #);
GL_MAX_MODELVIEW_STACK_DEPTH:      (# exit 0x0D36 #);
GL_MAX_NAME_STACK_DEPTH:           (# exit 0x0D37 #);
GL_MAX_PROJECTION_STACK_DEPTH:     (# exit 0x0D38 #);
GL_MAX_TEXTURE_STACK_DEPTH:        (# exit 0x0D39 #);
GL_MAX_EVAL_ORDER:                 (# exit 0x0D30 #);
GL_MAX_LIGHTS:                     (# exit 0x0D31 #);
GL_MAX_CLIP_PLANES:                (# exit 0x0D32 #);
GL_MAX_TEXTURE_SIZE:               (# exit 0x0D33 #);
GL_MAX_PIXEL_MAP_TABLE:            (# exit 0x0D34 #);
GL_MAX_VIEWPORT_DIMS:              (# exit 0x0D3A #);
GL_MAX_CLIENT_ATTRIB_STACK_DEPTH: (# exit 0x0D3B #);

(* Gets *)
GL_ATTRIB_STACK_DEPTH:             (# exit 0x0BB0 #);
GL_CLIENT_ATTRIB_STACK_DEPTH:      (# exit 0x0BB1 #);
GL_COLOR_CLEAR_VALUE:              (# exit 0x0C22 #);
GL_COLOR_WRITEMASK:                (# exit 0x0C23 #);
GL_CURRENT_INDEX:                  (# exit 0x0B01 #);
GL_CURRENT_COLOR:                  (# exit 0x0B00 #);
GL_CURRENT_NORMAL:                 (# exit 0x0B02 #);
GL_CURRENT_RASTER_COLOR:           (# exit 0x0B04 #);
GL_CURRENT_RASTER_DISTANCE:        (# exit 0x0B09 #);
GL_CURRENT_RASTER_INDEX:           (# exit 0x0B05 #);
GL_CURRENT_RASTER_POSITION:        (# exit 0x0B07 #);
```

```
GL_CURRENT_RASTER_TEXTURE_COORDS: (# exit 0x0B06 #);
GL_CURRENT_RASTER_POSITION_VALID: (# exit 0x0B08 #);
GL_CURRENT_TEXTURE_COORDS:       (# exit 0x0B03 #);
GL_INDEX_CLEAR_VALUE:            (# exit 0x0C20 #);
GL_INDEX_MODE:                   (# exit 0x0C30 #);
GL_INDEX_WRITEMASK:              (# exit 0x0C21 #);
GL_MODELVIEW_MATRIX:             (# exit 0x0BA6 #);
GL_MODELVIEW_STACK_DEPTH:        (# exit 0x0BA3 #);
GL_NAME_STACK_DEPTH:             (# exit 0x0D70 #);
GL_PROJECTION_MATRIX:            (# exit 0x0BA7 #);
GL_PROJECTION_STACK_DEPTH:       (# exit 0x0BA4 #);
GL_RENDER_MODE:                  (# exit 0x0C40 #);
GL_RGBA_MODE:                    (# exit 0x0C31 #);
GL_TEXTURE_MATRIX:               (# exit 0x0BA8 #);
GL_TEXTURE_STACK_DEPTH:          (# exit 0x0BA5 #);
GL_VIEWPORT:                     (# exit 0x0BA2 #);


(* Evaluators *)
GL_AUTO_NORMAL:                  (# exit 0x0D80 #);
GL_MAP1_COLOR_4:                 (# exit 0x0D90 #);
GL_MAP1_GRID_DOMAIN:             (# exit 0x0DD0 #);
GL_MAP1_GRID_SEGMENTS:           (# exit 0x0DD1 #);
GL_MAP1_INDEX:                   (# exit 0x0D91 #);
GL_MAP1_NORMAL:                  (# exit 0x0D92 #);
GL_MAP1_TEXTURE_COORD_1:         (# exit 0x0D93 #);
GL_MAP1_TEXTURE_COORD_2:         (# exit 0x0D94 #);
GL_MAP1_TEXTURE_COORD_3:         (# exit 0x0D95 #);
GL_MAP1_TEXTURE_COORD_4:         (# exit 0x0D96 #);
GL_MAP1_VERTEX_3:                (# exit 0x0D97 #);
GL_MAP1_VERTEX_4:                (# exit 0x0D98 #);
GL_MAP2_COLOR_4:                 (# exit 0x0DB0 #);
GL_MAP2_GRID_DOMAIN:             (# exit 0x0DD2 #);
GL_MAP2_GRID_SEGMENTS:           (# exit 0x0DD3 #);
GL_MAP2_INDEX:                   (# exit 0x0DB1 #);
GL_MAP2_NORMAL:                  (# exit 0x0DB2 #);
GL_MAP2_TEXTURE_COORD_1:         (# exit 0x0DB3 #);
GL_MAP2_TEXTURE_COORD_2:         (# exit 0x0DB4 #);
GL_MAP2_TEXTURE_COORD_3:         (# exit 0x0DB5 #);
GL_MAP2_TEXTURE_COORD_4:         (# exit 0x0DB6 #);
GL_MAP2_VERTEX_3:                (# exit 0x0DB7 #);
GL_MAP2_VERTEX_4:                (# exit 0x0DB8 #);
GL_COEFF:                        (# exit 0x0A00 #);
GL_DOMAIN:                       (# exit 0x0A02 #);
GL_ORDER:                        (# exit 0x0A01 #);

(* Hints *)
GL_FOG_HINT:                     (# exit 0x0C54 #);
GL_LINE_SMOOTH_HINT:             (# exit 0x0C52 #);
GL_PERSPECTIVE_CORRECTION_HINT:  (# exit 0x0C50 #);
GL_POINT_SMOOTH_HINT:            (# exit 0x0C51 #);
GL_POLYGON_SMOOTH_HINT:          (# exit 0x0C53 #);
GL_DONT_CARE:                    (# exit 0x1100 #);
GL_FASTEST:                      (# exit 0x1101 #);
GL_NICEST:                       (# exit 0x1102 #);

(* Scissor box *)
GL_SCISSOR_TEST:                 (# exit 0x0C11 #);
GL_SCISSOR_BOX:                  (# exit 0x0C10 #);

(* Pixel Mode / Transfer *)
GL_MAP_COLOR:                    (# exit 0x0D10 #);
GL_MAP_STENCIL:                  (# exit 0x0D11 #);
GL_INDEX_SHIFT:                  (# exit 0x0D12 #);
GL_INDEX_OFFSET:                 (# exit 0x0D13 #);
GL_RED_SCALE:                    (# exit 0x0D14 #);
```

```
GL_RED_BIAS:                    (# exit 0x0D15 #);
GL_GREEN_SCALE:                 (# exit 0x0D18 #);
GL_GREEN_BIAS:                  (# exit 0x0D19 #);
GL_BLUE_SCALE:                  (# exit 0x0D1A #);
GL_BLUE_BIAS:                   (# exit 0x0D1B #);
GL_ALPHA_SCALE:                 (# exit 0x0D1C #);
GL_ALPHA_BIAS:                  (# exit 0x0D1D #);
GL_DEPTH_SCALE:                 (# exit 0x0D1E #);
GL_DEPTH_BIAS:                  (# exit 0x0D1F #);
GL_PIXEL_MAP_S_TO_S_SIZE:       (# exit 0x0CB1 #);
GL_PIXEL_MAP_I_TO_I_SIZE:       (# exit 0x0CB0 #);
GL_PIXEL_MAP_I_TO_R_SIZE:       (# exit 0x0CB2 #);
GL_PIXEL_MAP_I_TO_G_SIZE:       (# exit 0x0CB3 #);
GL_PIXEL_MAP_I_TO_B_SIZE:       (# exit 0x0CB4 #);
GL_PIXEL_MAP_I_TO_A_SIZE:       (# exit 0x0CB5 #);
GL_PIXEL_MAP_R_TO_R_SIZE:       (# exit 0x0CB6 #);
GL_PIXEL_MAP_G_TO_G_SIZE:       (# exit 0x0CB7 #);
GL_PIXEL_MAP_B_TO_B_SIZE:       (# exit 0x0CB8 #);
GL_PIXEL_MAP_A_TO_A_SIZE:       (# exit 0x0CB9 #);
GL_PIXEL_MAP_S_TO_S:            (# exit 0x0C71 #);
GL_PIXEL_MAP_I_TO_I:            (# exit 0x0C70 #);
GL_PIXEL_MAP_I_TO_R:            (# exit 0x0C72 #);
GL_PIXEL_MAP_I_TO_G:            (# exit 0x0C73 #);
GL_PIXEL_MAP_I_TO_B:            (# exit 0x0C74 #);
GL_PIXEL_MAP_I_TO_A:            (# exit 0x0C75 #);
GL_PIXEL_MAP_R_TO_R:            (# exit 0x0C76 #);
GL_PIXEL_MAP_G_TO_G:            (# exit 0x0C77 #);
GL_PIXEL_MAP_B_TO_B:            (# exit 0x0C78 #);
GL_PIXEL_MAP_A_TO_A:            (# exit 0x0C79 #);
GL_PACK_ALIGNMENT:              (# exit 0x0D05 #);
GL_PACK_LSB_FIRST:              (# exit 0x0D01 #);
GL_PACK_ROW_LENGTH:             (# exit 0x0D02 #);
GL_PACK_SKIP_PIXELS:            (# exit 0x0D04 #);
GL_PACK_SKIP_ROWS:              (# exit 0x0D03 #);
GL_PACK_SWAP_BYTES:             (# exit 0x0D00 #);
GL_UNPACK_ALIGNMENT:            (# exit 0x0CF5 #);
GL_UNPACK_LSB_FIRST:            (# exit 0x0CF1 #);
GL_UNPACK_ROW_LENGTH:           (# exit 0x0CF2 #);
GL_UNPACK_SKIP_PIXELS:          (# exit 0x0CF4 #);
GL_UNPACK_SKIP_ROWS:            (# exit 0x0CF3 #);
GL_UNPACK_SWAP_BYTES:           (# exit 0x0CF0 #);
GL_ZOOM_X:                      (# exit 0x0D16 #);
GL_ZOOM_Y:                      (# exit 0x0D17 #);

(* Texture mapping *)
GL_TEXTURE_ENV:                 (# exit 0x2300 #);
GL_TEXTURE_ENV_MODE:            (# exit 0x2200 #);
GL_TEXTURE_1D:                  (# exit 0x0DE0 #);
GL_TEXTURE_2D:                  (# exit 0x0DE1 #);
GL_TEXTURE_WRAP_S:              (# exit 0x2802 #);
GL_TEXTURE_WRAP_T:              (# exit 0x2803 #);
GL_TEXTURE_MAG_FILTER:          (# exit 0x2800 #);
GL_TEXTURE_MIN_FILTER:          (# exit 0x2801 #);
GL_TEXTURE_ENV_COLOR:           (# exit 0x2201 #);
GL_TEXTURE_GEN_S:               (# exit 0x0C60 #);
GL_TEXTURE_GEN_T:               (# exit 0x0C61 #);
GL_TEXTURE_GEN_MODE:            (# exit 0x2500 #);
GL_TEXTURE_BORDER_COLOR:        (# exit 0x1004 #);
GL_TEXTURE_WIDTH:               (# exit 0x1000 #);
GL_TEXTURE_HEIGHT:              (# exit 0x1001 #);
GL_TEXTURE_BORDER:              (# exit 0x1005 #);
GL_TEXTURE_COMPONENTS:          (# exit 0x1003 #);
GL_TEXTURE_RED_SIZE:            (# exit 0x805C #);
GL_TEXTURE_GREEN_SIZE:          (# exit 0x805D #);
GL_TEXTURE_BLUE_SIZE:           (# exit 0x805E #);
GL_TEXTURE_ALPHA_SIZE:          (# exit 0x805F #);
```

```
GL_TEXTURE_LUMINANCE_SIZE:       (# exit 0x8060 #);
GL_TEXTURE_INTENSITY_SIZE:       (# exit 0x8061 #);
GL_NEAREST_MIPMAP_NEAREST:       (# exit 0x2700 #);
GL_NEAREST_MIPMAP_LINEAR:        (# exit 0x2702 #);
GL_LINEAR_MIPMAP_NEAREST:        (# exit 0x2701 #);
GL_LINEAR_MIPMAP_LINEAR:         (# exit 0x2703 #);
GL_OBJECT_LINEAR:                (# exit 0x2401 #);
GL_OBJECT_PLANE:                 (# exit 0x2501 #);
GL_EYE_LINEAR:                   (# exit 0x2400 #);
GL_EYE_PLANE:                    (# exit 0x2502 #);
GL_SPHERE_MAP:                   (# exit 0x2402 #);
GL_DECAL:                        (# exit 0x2101 #);
GL_MODULATE:                     (# exit 0x2100 #);
GL_NEAREST:                      (# exit 0x2600 #);
GL_REPEAT:                       (# exit 0x2901 #);
GL_CLAMP:                        (# exit 0x2900 #);
GL_S:                            (# exit 0x2000 #);
GL_T:                            (# exit 0x2001 #);
GL_R:                            (# exit 0x2002 #);
GL_Q:                            (# exit 0x2003 #);
GL_TEXTURE_GEN_R:                (# exit 0x0C62 #);
GL_TEXTURE_GEN_Q:                (# exit 0x0C63 #);

GL_PROXY_TEXTURE_1D:             (# exit 0x8063 #);
GL_PROXY_TEXTURE_2D:             (# exit 0x8064 #);
GL_TEXTURE_PRIORITY:             (# exit 0x8066 #);
GL_TEXTURE_RESIDENT:             (# exit 0x8067 #);
GL_TEXTURE_BINDING_1D:           (# exit 0x8068 #);
GL_TEXTURE_BINDING_2D:           (# exit 0x8069 #);

(* Internal texture formats *)
GL_ALPHA4:                       (# exit 0x803B #);
GL_ALPHA8:                       (# exit 0x803C #);
GL_ALPHA12:                      (# exit 0x803D #);
GL_ALPHA16:                      (# exit 0x803E #);
GL_LUMINANCE4:                   (# exit 0x803F #);
GL_LUMINANCE8:                   (# exit 0x8040 #);
GL_LUMINANCE12:                  (# exit 0x8041 #);
GL_LUMINANCE16:                  (# exit 0x8042 #);
GL_LUMINANCE4_ALPHA4:            (# exit 0x8043 #);
GL_LUMINANCE6_ALPHA2:            (# exit 0x8044 #);
GL_LUMINANCE8_ALPHA8:            (# exit 0x8045 #);
GL_LUMINANCE12_ALPHA4:           (# exit 0x8046 #);
GL_LUMINANCE12_ALPHA12:          (# exit 0x8047 #);
GL_LUMINANCE16_ALPHA16:          (# exit 0x8048 #);
GL_INTENSITY:                    (# exit 0x8049 #);
GL_INTENSITY4:                   (# exit 0x804A #);
GL_INTENSITY8:                   (# exit 0x804B #);
GL_INTENSITY12:                  (# exit 0x804C #);
GL_INTENSITY16:                  (# exit 0x804D #);
GL_R3_G3_B2:                     (# exit 0x2A10 #);
GL_RGB4:                         (# exit 0x804F #);
GL_RGB5:                         (# exit 0x8050 #);
GL_RGB8:                         (# exit 0x8051 #);
GL_RGB10:                        (# exit 0x8052 #);
GL_RGB12:                        (# exit 0x8053 #);
GL_RGB16:                        (# exit 0x8054 #);
GL_RGBA2:                        (# exit 0x8055 #);
GL_RGBA4:                        (# exit 0x8056 #);
GL_RGB5_A1:                      (# exit 0x8057 #);
GL_RGBA8:                        (# exit 0x8058 #);
GL_RGB10_A2:                     (# exit 0x8059 #);
GL_RGBA12:                       (# exit 0x805A #);
GL_RGBA16:                       (# exit 0x805B #);

(* Utility *)
```

```
GL_VENDOR:                      (# exit 0x1F00 #);
GL_RENDERER:                    (# exit 0x1F01 #);
GL_VERSION:                     (# exit 0x1F02 #);
GL_EXTENSIONS:                  (# exit 0x1F03 #);

(* Errors *)
GL_INVALID_VALUE:               (# exit 0x0501 #);
GL_INVALID_ENUM:                (# exit 0x0500 #);
GL_INVALID_OPERATION:           (# exit 0x0502 #);
GL_STACK_OVERFLOW:              (# exit 0x0503 #);
GL_STACK_UNDERFLOW:             (# exit 0x0504 #);
GL_OUT_OF_MEMORY:               (# exit 0x0505 #);

(*
 * 1.0 Extensions
 *)
(* GL_EXT_blend_minmax and GL_EXT_blend_color *)
GL_CONSTANT_COLOR_EXT:          (# exit 0x8001 #);
GL_ONE_MINUS_CONSTANT_COLOR_EXT: (# exit 0x8002 #);
GL_CONSTANT_ALPHA_EXT:          (# exit 0x8003 #);
GL_ONE_MINUS_CONSTANT_ALPHA_EXT: (# exit 0x8004 #);
GL_BLEND_EQUATION_EXT:          (# exit 0x8009 #);
GL_MIN_EXT:                     (# exit 0x8007 #);
GL_MAX_EXT:                     (# exit 0x8008 #);
GL_FUNC_ADD_EXT:                (# exit 0x8006 #);
GL_FUNC_SUBTRACT_EXT:           (# exit 0x800A #);
GL_FUNC_REVERSE_SUBTRACT_EXT:   (# exit 0x800B #);
GL_BLEND_COLOR_EXT:             (# exit 0x8005 #);

(* GL_EXT_polygon_offset *)
GL_POLYGON_OFFSET_EXT:          (# exit 0x8037 #);
GL_POLYGON_OFFSET_FACTOR_EXT:   (# exit 0x8038 #);
GL_POLYGON_OFFSET_BIAS_EXT:     (# exit 0x8039 #);

(* GL_EXT_vertex_array *)
GL_VERTEX_ARRAY_EXT:                    (# exit 0x8074 #);
GL_NORMAL_ARRAY_EXT:                    (# exit 0x8075 #);
GL_COLOR_ARRAY_EXT:                     (# exit 0x8076 #);
GL_INDEX_ARRAY_EXT:                     (# exit 0x8077 #);
GL_TEXTURE_COORD_ARRAY_EXT:             (# exit 0x8078 #);
GL_EDGE_FLAG_ARRAY_EXT:                 (# exit 0x8079 #);
GL_VERTEX_ARRAY_SIZE_EXT:               (# exit 0x807A #);
GL_VERTEX_ARRAY_TYPE_EXT:               (# exit 0x807B #);
GL_VERTEX_ARRAY_STRIDE_EXT:             (# exit 0x807C #);
GL_VERTEX_ARRAY_COUNT_EXT:              (# exit 0x807D #);
GL_NORMAL_ARRAY_TYPE_EXT:               (# exit 0x807E #);
GL_NORMAL_ARRAY_STRIDE_EXT:             (# exit 0x807F #);
GL_NORMAL_ARRAY_COUNT_EXT:              (# exit 0x8080 #);
GL_COLOR_ARRAY_SIZE_EXT:                (# exit 0x8081 #);
GL_COLOR_ARRAY_TYPE_EXT:                (# exit 0x8082 #);
GL_COLOR_ARRAY_STRIDE_EXT:              (# exit 0x8083 #);
GL_COLOR_ARRAY_COUNT_EXT:               (# exit 0x8084 #);
GL_INDEX_ARRAY_TYPE_EXT:                (# exit 0x8085 #);
GL_INDEX_ARRAY_STRIDE_EXT:              (# exit 0x8086 #);
GL_INDEX_ARRAY_COUNT_EXT:               (# exit 0x8087 #);
GL_TEXTURE_COORD_ARRAY_SIZE_EXT:        (# exit 0x8088 #);
GL_TEXTURE_COORD_ARRAY_TYPE_EXT:        (# exit 0x8089 #);
GL_TEXTURE_COORD_ARRAY_STRIDE_EXT:      (# exit 0x808A #);
GL_TEXTURE_COORD_ARRAY_COUNT_EXT:       (# exit 0x808B #);
GL_EDGE_FLAG_ARRAY_STRIDE_EXT:          (# exit 0x808C #);
GL_EDGE_FLAG_ARRAY_COUNT_EXT:           (# exit 0x808D #);
GL_VERTEX_ARRAY_POINTER_EXT:            (# exit 0x808E #);
GL_NORMAL_ARRAY_POINTER_EXT:            (# exit 0x808F #);
GL_COLOR_ARRAY_POINTER_EXT:             (# exit 0x8090 #);
GL_INDEX_ARRAY_POINTER_EXT:             (# exit 0x8091 #);
GL_TEXTURE_COORD_ARRAY_POINTER_EXT:     (# exit 0x8092 #);
```

```
GL_EDGE_FLAG_ARRAY_POINTER_EXT:          (# exit 0x8093 #);

(* GL_EXT_texture_object *)
GL_TEXTURE_PRIORITY_EXT:         (# exit 0x8066 #);
GL_TEXTURE_RESIDENT_EXT:         (# exit 0x8067 #);
GL_TEXTURE_1D_BINDING_EXT:       (# exit 0x8068 #);
GL_TEXTURE_2D_BINDING_EXT:       (# exit 0x8069 #);

(* GL_EXT_texture3D *)
GL_PACK_SKIP_IMAGES_EXT:         (# exit 0x806B #);
GL_PACK_IMAGE_HEIGHT_EXT:        (# exit 0x806C #);
GL_UNPACK_SKIP_IMAGES_EXT:       (# exit 0x806D #);
GL_UNPACK_IMAGE_HEIGHT_EXT:      (# exit 0x806E #);
GL_TEXTURE_3D_EXT:               (# exit 0x806F #);
GL_PROXY_TEXTURE_3D_EXT:         (# exit 0x8070 #);
GL_TEXTURE_DEPTH_EXT:            (# exit 0x8071 #);
GL_TEXTURE_WRAP_R_EXT:           (# exit 0x8072 #);
GL_MAX_3D_TEXTURE_SIZE_EXT:      (# exit 0x8073 #);
GL_TEXTURE_3D_BINDING_EXT:       (# exit 0x806A #);

(* GL_EXT_paletted_texture *)
GL_TABLE_TOO_LARGE_EXT:                  (# exit 0x8031 #);
GL_COLOR_TABLE_FORMAT_EXT:               (# exit 0x80D8 #);
GL_COLOR_TABLE_WIDTH_EXT:                (# exit 0x80D9 #);
GL_COLOR_TABLE_RED_SIZE_EXT:             (# exit 0x80DA #);
GL_COLOR_TABLE_GREEN_SIZE_EXT:           (# exit 0x80DB #);
GL_COLOR_TABLE_BLUE_SIZE_EXT:            (# exit 0x80DC #);
GL_COLOR_TABLE_ALPHA_SIZE_EXT:           (# exit 0x80DD #);
GL_COLOR_TABLE_LUMINANCE_SIZE_EXT:       (# exit 0x80DE #);
GL_COLOR_TABLE_INTENSITY_SIZE_EXT:       (# exit 0x80DF #);
GL_TEXTURE_INDEX_SIZE_EXT:               (# exit 0x80E1 #); (*XXX right value??*)
GL_COLOR_INDEX1_EXT:                     (# exit 0x80E2 #);
GL_COLOR_INDEX2_EXT:                     (# exit 0x80E3 #);
GL_COLOR_INDEX4_EXT:                     (# exit 0x80E4 #);
GL_COLOR_INDEX8_EXT:                     (# exit 0x80E5 #);
GL_COLOR_INDEX12_EXT:                    (# exit 0x80E6 #);
GL_COLOR_INDEX16_EXT:                    (# exit 0x80E7 #);

(* GL_EXT_shared_texture_palette *)
GL_SHARED_TEXTURE_PALETTE_EXT:           (# exit 0x81FB #);


(* GL_EXT_point_parameters *)
GL_POINT_SIZE_MIN_EXT:                   (# exit 0x8126 #);
GL_POINT_SIZE_MAX_EXT:                   (# exit 0x8127 #);
GL_POINT_FADE_THRESHOLD_SIZE_EXT:        (# exit 0x8128 #);
GL_DISTANCE_ATTENUATION_EXT:             (# exit 0x8129 #);


(* GL_NO_ERROR must be zero *)
GL_NO_ERROR: (# exit 0 #);

GL_CURRENT_BIT:          (# exit 0x00000001 #);
GL_POINT_BIT:            (# exit 0x00000002 #);
GL_LINE_BIT:             (# exit 0x00000004 #);
GL_POLYGON_BIT:          (# exit 0x00000008 #);
GL_POLYGON_STIPPLE_BIT:  (# exit 0x00000010 #);
GL_PIXEL_MODE_BIT:       (# exit 0x00000020 #);
GL_LIGHTING_BIT:         (# exit 0x00000040 #);
GL_FOG_BIT:              (# exit 0x00000080 #);
GL_DEPTH_BUFFER_BIT:     (# exit 0x00000100 #);
GL_ACCUM_BUFFER_BIT:     (# exit 0x00000200 #);
GL_STENCIL_BUFFER_BIT:   (# exit 0x00000400 #);
GL_VIEWPORT_BIT:         (# exit 0x00000800 #);
GL_TRANSFORM_BIT:        (# exit 0x00001000 #);
GL_ENABLE_BIT:           (# exit 0x00002000 #);
```

```
GL_COLOR_BUFFER_BIT:     (# exit 0x00004000 #);
GL_HINT_BIT:             (# exit 0x00008000 #);
GL_EVAL_BIT:             (# exit 0x00010000 #);
GL_LIST_BIT:             (# exit 0x00020000 #);
GL_TEXTURE_BIT:          (# exit 0x00040000 #);
GL_SCISSOR_BIT:          (# exit 0x00080000 #);
GL_ALL_ATTRIB_BITS:      (# exit 0x000fffff #);


GL_CLIENT_PIXEL_STORE_BIT:       (# exit 0x00000001 #);
GL_CLIENT_VERTEX_ARRAY_BIT:      (# exit 0x00000002 #);
GL_CLIENT_ALL_ATTRIB_BITS:       (# exit 0x0000FFFF #);


(*
 * Miscellaneous
 *)

glClearIndex: External (* Wrapped *)
  (# c: @real (* GLfloat *)
  enter (c)
  do 'wrapglClearIndex'−>CallC;
  #);

glClearColor: External
  (# red: @real (* GLclampf *);
     green: @real (* GLclampf *);
     blue: @real (* GLclampf *);
     alpha: @real (* GLclampf *)
  enter (red, green, blue, alpha)
  do 'wrapglClearColor'−>CallC;
  #);

glClear: External
  (# mask: @int32u (* GLbitfield *)
  enter (mask)
  do callStd
  #);

glIndexMask: External
  (# mask: @int32u (* GLuint *)
  enter (mask)
  do callStd
  #);

glColorMask: External
  (# red: @int8u (* GLboolean *);
     green: @int8u (* GLboolean *);
     blue: @int8u (* GLboolean *);
     alpha: @int8u (* GLboolean *)
  enter (red, green, blue, alpha)
  do callStd
  #);

glAlphaFunc: External
  (# func: @int32u (* GLenum *);
     ref: @real (* GLclampf *)
  enter (func, ref)
  do callStd
  #);

glBlendFunc: External
  (# sfactor: @integer (* GLenum *);
     dfactor: @integer (* GLenum *)
  enter (sfactor, dfactor)
  do callStd
```

```
  #);

glLogicOp: External
  (# opcode: @int32u (* GLenum *)
  enter (opcode)
  do callStd
  #);

glCullFace: External
  (# mode: @int32u (* GLenum *)
  enter (mode)
  do callStd
  #);

glFrontFace: External
  (# mode: @int32u (* GLenum *)
  enter (mode)
  do callStd
  #);

glPointSize: External (* wrapped *)
  (# size: @real (* GLfloat *);
  enter (size)
  do 'wrapglPointSize'−>CallC;
  #);

glLineWidth: External (* wrapped *)
  (# width: @real (* GLfloat *);
  enter (width)
  do 'wrapglLineWidth'−>CallC;
  #);

glLineStipple: External
  (# factor: @int32 (* GLint *);
     pattern: @int16u (* GLushort *);
  enter (factor, pattern)
  do callStd
  #);

glPolygonMode: External
  (# face: @int32u (* GLenum *);
     mode: @int32u (* GLenum *);
  enter (face, mode)
  do callStd
  #);

glPolygonOffset: External (* wrapped *) (* removed in gl1.2?*)
  (# factor: @real (* GLfloat *);
     units: @real (* GLfloat *);
  enter (factor, units)
  do 'wrapglPolygonOffset'−>CallC;
  #);

glPolygonStipple: External
  (# mask: @int8u (* GLubyte* *)
  enter (mask)
  do callStd
  #);

glGetPolygonStipple: External
  (# mask: @int8u (* GLubyte* *)
  enter (mask)
  do callStd
  #);

glEdgeFlag: External
```

```
  (# flag: @int8u (* GLboolean *)
  enter (flag)
  do callStd
  #);

glEdgeFlagv: External
  (# flag: @int32 (* GLboolean* *)
  enter (flag)
  do callStd
  #);

glScissor: External
  (# x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
     height: @int32 (* GLsizei *);
  enter (x, y, width, height)
  do callStd
  #);

glClipPlane: External
  (# plane: @int32u (* GLenum *);
     equation: @int32 (* GLdouble* *);
  enter (plane, equation)
  do callStd
  #);

glGetClipPlane: External
  (# plane: @int32u (* GLenum *);
     equation: @int32 (* GLdouble* *)
  enter (plane, equation)
  do callStd
  #);

glDrawBuffer: External
  (# mode: @int32u (* GLenum *);
  enter (mode)
  do callStd
  #);

glReadBuffer: External
  (# mode: @int32u (* GLenum *);
  enter (mode)
  do callStd
  #);

glEnable: External
  (# cap: @int32u (* GLenum *)
  enter (cap)
  do callStd
  #);

glDisable: External
  (# cap: @int32u (* GLenum *)
  enter (cap)
  do callStd
  #);

glIsEnabled: External
  (# cap: @int32u (* GLenum *);
     result: @int8u (* GLboolean *);
  enter cap
  do callStd
  exit result
  #);
```

```
glEnableClientState: External
  (# cap: @int32u (* GLenum *)
  enter (cap)
  do callStd
  #);


glDisableClientState: External
  (# cap: @int32u (* GLenum *)
  enter (cap)
  do callStd
  #);

glGetBooleanv: External
  (# pname: @int32u (* GLenum *);
     params: @int32 (* GLboolean* *);
  enter (pname, params)
  do callStd
  #);

glGetDoublev: External
  (# pname: @int32u (* GLenum *);
     params: @int32 (* GLdouble* *);
  enter (pname, params)
  do callStd
  #);

glGetFloatv: External
  (# pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *);
  enter (pname, params)
  do callStd
  #);

glGetIntegerv: External
  (# pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *);
  enter (pname, params)
  do callStd
  #);


glPushAttrib: External
  (# mask: @int32u (* GLbitfield *)
  enter (mask)
  do callStd
  #);

glPopAttrib: External
  (#
  do callStd
  #);


glPushClientAttrib: External
  (# mask: @int32u (* GLbitfield *)
  enter (mask)
  do callStd
  #);

glPopClientAttrib: External
  (#
  do callStd
  #);
```

```
glRenderMode: External
  (# mode: @int32u (* GLenum *);
     result: @int32 (* GLint *);
  enter mode
  do callStd;
  exit result
  #);

glGetError: External
  (# error: @int32u (* GLenum *);
  do callStd;
  exit error
  #);

glGetString: External
  (# name: @int32 (* GLubyte* *);
     result: [0]@char (* GLubyte* *);
  enter name
  do callStd;
  exit result
  #);

glFinish: External
  (#
  do callStd;
  #);

glFlush: External
  (#
  do callStd;
  #);

glHint: External
  (# target: @int32u (* GLenum *);
     mode: @int32u (* GLenum *);
  enter (target, mode)
  do callStd;
  #);



(*
 * Depth Buffer
 *)

glClearDepth: External
  (# depth: @real (* GLclampd *)
  enter (depth)
  do callStd;
  #);

glDepthFunc: External
  (# func: @int32u (* GLenum *)
  enter (func)
  do callStd;
  #);

glDepthMask: External
  (# flag: @int8u (* GLboolean *)
  enter (flag)
  do callStd;
  #);

glDepthRange: External
  (# near_val: @real (* GLclampd *);
```

```
      far_val: @real (* GLclampd *);
  enter (near_val, far_val)
  do callStd;
  #);


(*
 * Accumulation Buffer
 *)

glClearAccum: External
  (# red: @real (* GLfloat *);
     green: @real (* GLfloat *);
     blue: @real (* GLfloat *);
     alpha: @real (* GLfloat *);
  enter (red, green, blue, alpha)
  do callStd;
  #);

glAccum: External (* wrapped *)
  (# op: @int32u (* GLenum *);
     value: @real (* GLfloat *);
  enter (op, value)
  do 'wrapglAccum'−>CallC;
  #);



(*
 * Transformation
 *)

glMatrixMode: External
  (# mode: @int32u (* GLenum *)
  enter (mode)
  do callStd;
  #);

glOrtho: External
  (# left: @real (* GLdouble *);
     right: @real (* GLdouble *);
     bottom: @real (* GLdouble *);
     top: @real (* GLdouble *);
     near_val: @real (* GLdouble *);
     far_val: @real (* GLdouble *)
  enter (left, right, bottom, top, near_val, far_val)
  do callStd;
  #);

glFrustum: External
  (# left: @real (* GLdouble *);
     right: @real (* GLdouble *);
     bottom: @real (* GLdouble *);
     top: @real (* GLdouble *);
     near_val: @real (* GLdouble *);
     far_val: @real (* GLdouble *);
  enter (left, right, bottom, top, near_val, far_val)
  do callStd;
  #);

glViewport: External
  (# x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
     height: @int32 (* GLsizei *);
  enter (x, y, width, height)
```

```
  do callStd
  #);

glPushMatrix: External
  (#
  do callStd;
  #);

glPopMatrix: External
  (#
  do callStd;
  #);

glLoadIdentity: External
  (#
  do callStd;
  #);

glLoadMatrixd: External
  (# m: @int32 (* GLdouble* *)
  enter (m)
  do callStd;
  #);

glLoadMatrixf: External
  (# m: @int32 (* GLfloat* *)
  enter (m)
  do callStd;
  #);

glMultMatrixd: External
  (# m: @int32 (* GLdouble* *)
  enter (m)
  do callStd;
  #);

glMultMatrixf: External
  (# m: @int32 (* GLfloat* *)
  enter (m)
  do callStd;
  #);

glRotated: External
  (# angle: @real (* GLdouble *);
     x: @real (* GLdouble *);
     y: @real (* GLdouble *);
     z: @real (* GLdouble *);
  enter (angle, x, y, z)
  do callStd;
  #);

glRotatef:glRotated(# #);

glScaled: External
  (# x: @real (* GLdouble *);
     y: @real (* GLdouble *);
     z: @real (* GLdouble *);
  enter (x, y, z)
  do callStd;
  #);

glScalef:glScaled(# #);

glTranslated: External
  (# x: @real (* GLdouble *);
     y: @real (* GLdouble *);
```

```
      z: @real (* GLdouble *);
  enter (x, y, z)
  do callStd;
  #);

glTranslatef:glTranslated(# #);

(*
 * Display Lists
 *)

glIsList: External
  (#  list: @int32 (* GLuint *);
      result: @int8u (* GLboolean *);
  enter list
  do callStd;
  exit result
  #);

glDeleteLists: External
  (# list: @int32 (* GLuint *);
      range: @int32 (* GLsizei *);
  enter (list, range)
  do callStd;
  #);

glGenLists: External
  (# range: @int32 (* GLsizei *);
      result: @int32u (* GLuint *);
  enter range
  do callStd;
  exit result
  #);

glNewList: External
  (# list: @int32 (* GLuint *);
      mode: @int32u (* GLenum *);
  enter (list, mode)
  do callStd;
  #);

glEndList: External
  (#
  do callStd;
  #);

glCallList: External
  (# list: @int32 (* GLuint *);
  enter (list)
  do callStd;
  #);

glCallLists: External
  (# n: @int32 (* GLsizei *);
      type: @int32u (* GLenum *);
      lists: @int32 (* GLvoid* *);
  enter (n, type, lists)
  do callStd;
  #);

glListBase: External
  (# base: @int32u (* GLuint *)
  enter (base)
  do callStd;
  #);
```

```
(*
 * Drawing Functions
 *)

glBegin: External
  (# mode: @int32u (* GLenum *)
  enter (mode)
  do callStd;
  #);

glEnd: External
  (#
  do callStd;
  #);


glVertex2d: External
  (# x: @real (* GLdouble *);
     y: @real (* GLdouble *);
  enter (x, y)
  do callStd;
  #);

glVertex2f:glVertex2d(# #);

glVertex2i: External
  (# x: @int32 (* GLint *);
     y: @int32 (* GLint *);
  enter (x, y)
  do callStd;
  #);

glVertex2s: External
  (# x: @int16 (* GLshort *);
     y: @int16 (* GLshort *);
  enter (x, y)
  do callStd;
  #);

glVertex3d: External
  (# x: @real (* GLdouble *);
     y: @real (* GLdouble *);
     z: @real (* GLdouble *);
  enter (x, y, z)
  do callStd;
  #);

glVertex3f:glVertex3d(# #);

glVertex3i: External
  (# x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     z: @int32 (* GLint *);
  enter (x, y, z)
  do callStd;
  #);

glVertex3s: External
  (# x: @int16 (* GLshort *);
     y: @int16 (* GLshort *);
     z: @int16 (* GLshort *);
  enter (x, y, z)
  do callStd;
  #);
```

```
glVertex4d: External
  (# x: @real (* GLdouble *);
     y: @real (* GLdouble *);
     z: @real (* GLdouble *);
     w: @real (* GLdouble *);
  enter (x, y, z, w)
  do callStd;
  #);

glVertex4f: glVertex4d(# #);

glVertex4i: External
  (# x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     z: @int32 (* GLint *);
     w: @int32 (* GLint *);
  enter (x, y, z, w)
  do callStd;
  #);

glVertex4s: External
  (# x: @int16 (* GLshort *);
     y: @int16 (* GLshort *);
     z: @int16 (* GLshort *);
     w: @int16 (* GLshort *);
  enter (x, y, z, w)
  do callStd;
  #);

glVertex2dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glVertex2fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);

glVertex2iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);

glVertex2sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);

glVertex3dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glVertex3fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);
```

```
glVertex3iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);

glVertex3sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);

glVertex4dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glVertex4fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);

glVertex4iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);

glVertex4sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);


glNormal3b: External
  (# nx: @int8 (* GLbyte *);
     ny: @int8 (* GLbyte *);
     nz: @int8 (* GLbyte *);
  enter (nx, ny, nz)
  do callStd;
  #);

glNormal3d: External
  (# nx: @real (* GLdouble *);
     ny: @real (* GLdouble *);
     nz: @real (* GLdouble *);
  enter (nx, ny, nz)
  do callStd;
  #);

glNormal3f: glNormal3d(# #);

glNormal3i: External
  (# nx: @int32 (* GLint *);
     ny: @int32 (* GLint *);
     nz: @int32 (* GLint *);
  enter (nx, ny, nz)
  do callStd;
  #);

glNormal3s: External
  (# nx: @int16 (* GLshort *);
     ny: @int16 (* GLshort *);
```

```
     nz: @int16 (* GLshort *);
  enter (nx, ny, nz)
  do callStd;
  #);

glNormal3bv: External
  (# v: @int32 (* GLbyte* *)
  enter (v)
  do callStd;
  #);

glNormal3dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glNormal3fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);

glNormal3iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);

glNormal3sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);


glIndexd: External
  (# c: @real (* GLdouble *)
  enter (c)
  do callStd;
  #);

glIndexf: glIndexd(# #);

glIndexi: External
  (# c: @int32 (* GLint *);
  enter (c)
  do callStd;
  #);

glIndexs: External
  (# c: @int16 (* GLshort *);
  enter (c)
  do callStd;
  #);

glIndexub: External
  (# c: @int8u (* GLubyte *);
  enter (c)
  do callStd;
  #);

glIndexdv: External
  (# c: @int32 (* GLdouble* *);
  enter (c)
  do callStd;
```

```
  #);

glIndexfv: External
  (# c: @int32 (* GLfloat* *);
  enter (c)
  do callStd;
  #);

glIndexiv: External
  (# c: @int32 (* GLint* *);
  enter (c)
  do callStd;
  #);

glIndexsv: External
  (# c: @int32 (* GLshort* *);
  enter (c)
  do callStd;
  #);

glIndexubv: External
  (# c: @int32 (* GLubyte* *);
  enter (c)
  do callStd;
  #);  (* 1.1 *)

glColor3b: External
  (# red: @int8 (* GLbyte *);
     green: @int8 (* GLbyte *);
     blue: @int8 (* GLbyte *);
  enter (red, green, blue)
  do callStd;
  #);

glColor3d: External
  (# red: @real (* GLdouble *);
     green: @real (* GLdouble *);
     blue: @real (* GLdouble *);
  enter (red, green, blue)
  do callStd;
  #);

glColor3f: glColor3d(# #);

glColor3i: External
  (# red: @int32 (* GLint *);
     green: @int32 (* GLint *);
     blue: @int32 (* GLint *);
  enter (red, green, blue)
  do callStd;
  #);

glColor3s: External
  (# red: @int16 (* GLshort *);
     green: @int16 (* GLshort *);
     blue: @int16 (* GLshort *);
  enter (red, green, blue)
  do callStd;
  #);

glColor3ub: External
  (# red: @int8u (* GLubyte *);
     green: @int8u (* GLubyte *);
     blue: @int8u (* GLubyte *);
  enter (red, green, blue)
  do callStd;
```

```
  #);

glColor3ui: External
  (# red: @int32 (* GLuint *);
     green: @int32 (* GLuint *);
     blue: @int32 (* GLuint *);
  enter (red, green, blue)
  do callStd;
  #);

glColor3us: External
  (# red: @int16u (* GLushort *);
     green: @int16u (* GLushort *);
     blue: @int16u (* GLushort *);
  enter (red, green, blue)
  do callStd;
  #);

glColor4b: External
  (# red: @int8 (* GLbyte *);
     green: @int8 (* GLbyte *);
     blue: @int8 (* GLbyte *);
     alpha: @int8 (* GLbyte *);
  enter (red, green, blue, alpha)
  do callStd;
  #);

glColor4f: External (* Wrapped *)
  (# red: @real (* GLfloat *);
     green: @real (* GLfloat *);
     blue: @real (* GLfloat *);
     alpha: @real (* GLfloat *);
  enter (red, green, blue, alpha)
  do 'wrapglColor4f'->CallC;
  #);

glColor4d: External
  (# red: @real (* GLdouble *);
     green: @real (* GLdouble *);
     blue: @real (* GLdouble *);
     alpha: @real (* GLdouble *);
  enter (red, green, blue, alpha)
  do callStd;
  #);;

glColor4i: External
  (# red: @int32 (* GLint *);
     green: @int32 (* GLint *);
     blue: @int32 (* GLint *);
     alpha: @int32 (* GLint *);
  enter (red, green, blue, alpha)
  do callStd;
  #);

glColor4s: External
  (# red: @int16 (* GLshort *);
     green: @int16 (* GLshort *);
     blue: @int16 (* GLshort *);
     alpha: @int16 (* GLshort *);
  enter (red, green, blue, alpha)
  do callStd;
  #);

glColor4ub: External
  (# red: @int8u (* GLubyte *);
     green: @int8u (* GLubyte *);
```

```
      blue: @int8u (* GLubyte *);
      alpha: @int8u (* GLubyte *);
  enter (red, green, blue, alpha)
  do callStd;
  #);

glColor4ui: External
  (# red: @int32 (* GLuint *);
     green: @int32 (* GLuint *);
     blue: @int32 (* GLuint *);
     alpha: @int32 (* GLuint *);
  enter (red, green, blue, alpha)
  do callStd;
  #);

glColor4us: External
  (# red: @int16u (* GLushort *);
     green: @int16u (* GLushort *);
     blue: @int16u (* GLushort *);
     alpha: @int16u (* GLushort *);
  enter (red, green, blue, alpha)
  do callStd;
  #);

glColor3bv: External
  (# v: @int32 (* GLbyte* *)
  enter (v)
  do callStd;
  #);

glColor3dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glColor3fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);

glColor3iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);

glColor3sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);

glColor3ubv: External
  (# v: @int8u (* GLubyte* *)
  enter (v)
  do callStd;
  #);

glColor3uiv: External
  (# v: @int32 (* GLuint* *)
  enter (v)
  do callStd;
  #);
```

```
glColor3usv: External
  (# v: @int32 (* GLushort* *)
  enter (v)
  do callStd;
  #);

glColor4bv: External
  (# v: @int32 (* GLbyte* *)
  enter (v)
  do callStd;
  #);

glColor4dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glColor4fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  #);

glColor4iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);

glColor4sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);

glColor4ubv: External
  (# v: @int8u (* GLubyte* *)
  enter (v)
  do callStd;
  #);

glColor4uiv: External
  (# v: @int32 (* GLuint* *)
  enter (v)
  do callStd;
  #);

glColor4usv: External
  (# v: @int32 (* GLushort* *)
  enter (v)
  do callStd;
  #);


glTexCoord1d: External
  (# s: @real (* GLdouble *)
  enter (s)
  do callStd;
  #);

glTexCoord1f: External (* Wrapped *)
  (# s: @real (* GLfloat *)
  enter (s)
  do 'wrapglTexCoord1f'->CallC;
  #);
```

```
glTexCoord1i: External
  (# s: @int32 (* GLint *)
  enter (s)
  do callStd;
  #);

glTexCoord1s: External
  (# s: @int16 (* GLshort *)
  enter (s)
  do callStd;
  #);

glTexCoord2d: External
  (# s: @real (* GLdouble *);
     t: @real (* GLdouble *);
  enter (s, t)
  do callStd;
  #);

glTexCoord2f: glTexCoord2d(# #);

glTexCoord2i: External
  (# s: @int32 (* GLint *);
     t: @int32 (* GLint *);
  enter (s, t)
  do callStd;
  #);

glTexCoord2s: External
  (# s: @int16 (* GLshort *);
     t: @int16 (* GLshort *);
  enter (s, t)
  do callStd;
  #);

glTexCoord3d: External
  (# s: @real (* GLdouble *);
     t: @real (* GLdouble *);
     r: @real (* GLdouble *);
  enter (s, t, r)
  do callStd;
  #);

glTexCoord3f: glTexCoord3d(# #);

glTexCoord3i: External
  (# s: @int32 (* GLint *);
     t: @int32 (* GLint *);
     r: @int32 (* GLint *);
  enter (s, t, r)
  do callStd;
  #);

glTexCoord3s: External
  (# s: @int16 (* GLshort *);
     t: @int16 (* GLshort *);
     r: @int16 (* GLshort *);
  enter (s, t, r)
  do callStd;
  #);

glTexCoord4d: External
  (# s: @real (* GLdouble *);
     t: @real (* GLdouble *);
     r: @real (* GLdouble *);
     q: @real (* GLdouble *);
```

```
  enter (s, t, r, q)
  do callStd;
  #);

glTexCoord4f: glTexCoord4d(# #);

glTexCoord4i: External
  (# s: @int32 (* GLint *);
     t: @int32 (* GLint *);
     r: @int32 (* GLint *);
     q: @int32 (* GLint *);
  enter (s, t, r, q)
  do callStd;
  #);

glTexCoord4s: External
  (# s: @int16 (* GLshort *);
     t: @int16 (* GLshort *);
     r: @int16 (* GLshort *);
     q: @int16 (* GLshort *);
  enter (s, t, r, q)
  do callStd;
  #);

glTexCoord1dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glTexCoord1fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);

glTexCoord1iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);

glTexCoord1sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);

glTexCoord2dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glTexCoord2fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);

glTexCoord2iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);
```

```
glTexCoord2sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);

glTexCoord3dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glTexCoord3fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);

glTexCoord3iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);

glTexCoord3sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);

glTexCoord4dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glTexCoord4fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);

glTexCoord4iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);

glTexCoord4sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);


glRasterPos2d: External
  (# x: @real (* GLdouble *);
     y: @real (* GLdouble *);
  enter (x, y)
  do callStd;
  #);

glRasterPos2f: glRasterPos2d(# #);

glRasterPos2i: External
  (# x: @int32 (* GLint *);
```

```
     y: @int32 (* GLint *);
  enter (x, y)
  do callStd;
  #);

glRasterPos2s: External
  (# x: @int16 (* GLshort *);
     y: @int16 (* GLshort *);
  enter (x, y)
  do callStd;
  #);

glRasterPos3d: External
  (# x: @real (* GLdouble *);
     y: @real (* GLdouble *);
     z: @real (* GLdouble *);
  enter (x, y, z)
  do callStd;
  #);

glRasterPos3f: glRasterPos3d(# #);

glRasterPos3i: External
  (# x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     z: @int32 (* GLint *);
  enter (x, y, z)
  do callStd;
  #);

glRasterPos3s: External
  (# x: @int16 (* GLshort *);
     y: @int16 (* GLshort *);
     z: @int16 (* GLshort *);
  enter (x, y, z)
  do callStd;
  #);

glRasterPos4d: External
  (# x: @real (* GLdouble *);
     y: @real (* GLdouble *);
     z: @real (* GLdouble *);
     w: @real (* GLdouble *);
  enter (x, y, z, w)
  do callStd;
  #);

glRasterPos4f: glRasterPos4d(# #);

glRasterPos4i: External
  (# x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     z: @int32 (* GLint *);
     w: @int32 (* GLint *);
  enter (x, y, z, w)
  do callStd;
  #);

glRasterPos4s: External
  (# x: @int16 (* GLshort *);
     y: @int16 (* GLshort *);
     z: @int16 (* GLshort *);
     w: @int16 (* GLshort *);
  enter (x, y, z, w)
  do callStd;
  #);
```

```
glRasterPos2dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glRasterPos2fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);

glRasterPos2iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);

glRasterPos2sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);

glRasterPos3dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glRasterPos3fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);

glRasterPos3iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);

glRasterPos3sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);

glRasterPos4dv: External
  (# v: @int32 (* GLdouble* *)
  enter (v)
  do callStd;
  #);

glRasterPos4fv: External
  (# v: @int32 (* GLfloat* *)
  enter (v)
  do callStd;
  #);

glRasterPos4iv: External
  (# v: @int32 (* GLint* *)
  enter (v)
  do callStd;
  #);
```

```
glRasterPos4sv: External
  (# v: @int32 (* GLshort* *)
  enter (v)
  do callStd;
  #);


glRectd: External
  (# x1: @real (* GLdouble *);
     y1: @real (* GLdouble *);
     x2: @real (* GLdouble *);
     y2: @real (* GLdouble *);
  enter (x1, y1, x2, y2)
  do callStd;
  #);

glRectf: glRectd(# #);

glRecti: External
  (# x1: @int32 (* GLint *);
     y1: @int32 (* GLint *);
     x2: @int32 (* GLint *);
     y2: @int32 (* GLint *);
  enter (x1, y1, x2, y2)
  do callStd;
  #);

glRects: External
  (# x1: @int16 (* GLshort *);
     y1: @int16 (* GLshort *);
     x2: @int16 (* GLshort *);
     y2: @int16 (* GLshort *);
  enter (x1, y1, x2, y2)
  do callStd;
  #);

glRectdv: External
  (# v1: @int32 (* GLdouble* *); v2: @int32 (* GLdouble* *)
  enter (v1, v2)
  do callStd;
  #);

glRectfv: External
  (# v1: @int32 (* GLfloat* *); v2: @int32 (* GLfloat* *)
  enter (v1, v2)
  do callStd;
  #);

glRectiv: External
  (# v1: @int32 (* GLint* *); v2: @int32 (* GLint* *)
  enter (v1, v2)
  do callStd;
  #);

glRectsv: External
  (# v1: @int32 (* GLshort* *); v2: @int32 (* GLshort* *)
  enter (v1,v2)
  #);



(*
 * Vertex Arrays  (1.1)
 *)
```

```
glVertexPointer: External
  (# size: @int32 (* GLint *);
     type: @int32u (* GLenum *);
     stride: @int32 (* GLsizei *);
     ptr: @int32 (* GLvoid* *);
  enter (size, type, stride, ptr)
  do callStd;
  #);

glNormalPointer: External
  (# type: @int32u (* GLenum *);
     stride: @int32 (* GLsizei *);
     ptr: @int32 (* GLvoid* *);
  enter (type, stride, ptr)
  do callStd;
  #);

glColorPointer: External
  (# size: @int32 (* GLint *);
     type: @int32u (* GLenum *);
     stride: @int32 (* GLsizei *);
     ptr: @int32 (* GLvoid* *)
  enter (size, type, stride, ptr)
  do callStd;
  #);

glIndexPointer: External
  (# type: @int32u (* GLenum *);
     stride: @int32 (* GLsizei *);
     ptr: @int32 (* GLvoid* *);
  enter (type, stride, ptr)
  do callStd;
  #);

glTexCoordPointer: External
  (# size: @int32 (* GLint *);
     type: @int32u (* GLenum *);
     stride: @int32 (* GLsizei *);
     ptr: @int32 (* GLvoid* *);
  enter (size, type, stride, ptr)
  do callStd;
  #);

glEdgeFlagPointer: External
  (# stride: @int32 (* GLsizei *);
     ptr: @int32 (* GLboolean* *);
  enter (stride, ptr)
  do callStd;
  #);

glGetPointerv: External
  (# pname: @int32u (* GLenum *);
     params: @int32 (* void** *);
  enter (pname, params)
  do callStd;
  #);

glArrayElement: External
  (# i: @int32 (* GLint *)
  enter (i)
  do callStd;
  #);

glDrawArrays: External
  (# mode: @int32u (* GLenum *);
     first: @int32 (* GLint *);
```

```
     count: @int32 (* GLsizei *);
  enter (mode, first, count)
  do callStd;
  #);

glDrawElements: External
  (# mode: @int32u (* GLenum *);
     count: @int32 (* GLsizei *);
     type: @int32u (* GLenum *);
     indices: @int32 (* GLvoid* *);
  enter (mode, count, type, indices)
  do callStd;
  #);

glInterleavedArrays: External
  (# format: @int32u (* GLenum *);
     stride: @int32 (* GLsizei *);
     pointer: @int32 (* GLvoid* *);
  enter (format, stride, pointer)
  do callStd;
  #);


(*
 * Lighting
 *)

glShadeModel: External
  (# mode: @int32u (* GLenum *)
  enter (mode)
  do callStd;
  #);

glLightf: External (* Wrapped *)
  (# light: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     param: @real (* GLfloat *);
  enter (light, pname, param)
  do 'wrapglLightf'−>CallC;
  #);

glLighti: External
  (# light: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     param: @int32 (* GLint *);
  enter (light, pname, param)
  do callStd;
  #);

glLightfv: External
  (# light: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *);
  enter (light, pname, params)
  do 'wrapglLightfv'−>CallC;
  #);

glLightiv: External
  (# light: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *);
  enter (light, pname, params)
  do callStd;
  #);

glGetLightfv: External
```

```
  (# light: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *);
  enter (light, pname, params)
  do callStd;
  #);

glGetLightiv: External
  (# light: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *);
  enter (light, pname, params)
  do callStd;
  #);

glLightModelf: External (* wrapped *)
  (# pname: @int32u (* GLenum *);
     param: @real (* GLfloat *);
  enter (pname, param)
  do 'wrapglLightModelf'->CallC;
  #);

glLightModeli: External
  (# pname: @int32u (* GLenum *);
     param: @int32 (* GLint *);
  enter (pname, param)
  do callStd;
  #);

glLightModelfv: External (* wrapped *)
  (# pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *);
  enter (pname, params)
  do 'wrapglLightModelfv'->CallC;
  #);

glLightModeliv: External
  (# pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *);
  enter (pname, params)
  do callStd;
  #);

glMaterialf: External (* wrapped *)
  (# face: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     param: @real (* GLfloat *);
  enter (face, pname, param)
  do 'wrapglMaterialf'->CallC;
  #);

glMateriali: External
  (# face: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     param: @int32 (* GLint *);
  enter (face, pname, param)
  do callStd;
  #);

glMaterialfv: External (* wrapped *)
  (# face: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *);
  enter (face, pname, params)
  do 'wrapglMaterialfv'->CallC;
  #);
```

```
glMaterialiv: External
  (# face: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *);
  enter (face, pname, params)
  do callStd;
  #);

glGetMaterialfv: External
  (# face: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *);
  enter (face, pname, params)
  do callStd;
  #);

glGetMaterialiv: External
  (# face: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *);
  enter (face, pname, params)
  do callStd;
  #);

glColorMaterial: External
  (# face: @int32u (* GLenum *);
     mode: @int32u (* GLenum *);
  enter (face, mode)
  do callStd;
  #);




(*
 * Raster functions
 *)

glPixelZoom: External (* wrapped *)
  (# xfactor: @real (* GLfloat *);
     yfactor: @real (* GLfloat *);
  enter (xfactor, yfactor)
  do 'wrapglPixelZoom'−>CallC;
  #);

glPixelStoref: External (* wrapped *)
  (# pname: @int32u (* GLenum *);
     param: @real (* GLfloat *);
  enter (pname, param)
  do 'wrapglPixelStoref'−>CallC;
  #);

glPixelStorei: External
  (# pname: @int32u (* GLenum *);
     param: @int32 (* GLint *);
  enter (pname, param)
  do callStd;
  #);

glPixelTransferf: External (* wrapped *)
  (# pname: @int32u (* GLenum *);
     param: @real (* GLfloat *);
  enter (pname, param)
  do 'wrapglPixelTransferf'−>CallC;
  #);
```

```
glPixelTransferi: External
  (# pname: @int32u (* GLenum *); param: @int32 (* GLint *)
  enter (pname, param)
  do callStd;
  #);

glPixelMapfv: External
  (# map: @int32u (* GLenum *);
     mapsize: @int32 (* GLint *);
     values: @int32 (* GLfloat* *)
  enter (map, mapsize, values)
  do callStd;
  #);

glPixelMapuiv: External
  (# map: @int32u (* GLenum *);
     mapsize: @int32 (* GLint *);
     values: @int32 (* GLuint* *);
  enter (map, mapsize, values)
  do callStd;
  #);

glPixelMapusv: External
  (# map: @int32u (* GLenum *);
     mapsize: @int32 (* GLint *);
     values: @int32 (* GLushort* *);
  enter (map, mapsize, values)
  do callStd;
  #);

glGetPixelMapfv: External
  (# map: @int32u (* GLenum *);
     values: @int32 (* GLfloat* *);
  enter (map, values)
  do callStd;
  #);

glGetPixelMapuiv: External
  (# map, values: @int32 (* GLuint* *)
  enter (map, values)
  do callStd;
  #);

glGetPixelMapusv: External
  (# map: @int32u (* GLenum *);
     values: @int32 (* GLushort* *);
  enter (map, values)
  do callStd;
  #);

glBitmap: External
  (# width: @int32 (* GLsizei *);
     height: @int32 (* GLsizei *);
     xorig: @real (* GLfloat *);
     yorig: @real (* GLfloat *);
     xmove: @real (* GLfloat *);
     ymove: @real (* GLfloat *);
     bitmap: @int32 (* GLubyte* *);
  enter (width, height, xorig, yorig, xmove, ymove, bitmap)
  do callStd;
  #);

glReadPixels: External
  (# x: @int32 (* GLint *);
     y: @int32 (* GLint *);
```

```
      width: @int32 (* GLsizei *);
      height: @int32 (* GLsizei *);
      format: @int32u (* GLenum *);
      type: @int32u (* GLenum *);
      pixels: @int32 (* GLvoid* *);
  enter (x, y, width, height, format, type, pixels)
  do callStd;
  #);

glDrawPixels: External
  (# width: @int32 (* GLsizei *);
     height: @int32 (* GLsizei *);
     format: @int32u (* GLenum *);
     type: @int32u (* GLenum *);
     pixels: @int32 (* GLvoid* *)
  enter (width, height, format, type, pixels)
  do callStd;
  #);

glCopyPixels: External
  (# x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
     height: @int32 (* GLsizei *);
     type: @int32u (* GLenum *);
  enter (x, y, width, height, type)
  do callStd;
  #);



(*
 * Stenciling
 *)

glStencilFunc: External
  (# func: @int32u (* GLenum *);
     ref: @int32 (* GLint *);
     mask: @int32 (* GLuint *);
  enter (func, ref, mask)
  do callStd;
  #);

glStencilMask: External
  (# mask: @int32 (* GLuint *);
  enter (mask)
  do callStd;
  #);

glStencilOp: External
  (# fail: @int32u (* GLenum *);
     zfail: @int32u (* GLenum *);
     zpass: @int32u (* GLenum *);
  enter (fail, zfail, zpass)
  do callStd;
  #);

glClearStencil: External
  (# s: @int32 (* GLint *);
  enter (s)
  do callStd;
  #);



(*
```

```
 * Texture mapping
 *)

glTexGend: External
  (# coord: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     param: @real (* GLdouble *);
  enter (coord, pname, param)
  do callStd;
  #);

glTexGenf: glTexGend(# #);

glTexGeni: External
  (# coord: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     param: @int32 (* GLint *);
  enter (coord, pname, param)
  do callStd;
  #);

glTexGendv: External
  (# coord: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLdouble* *);
  enter (coord, pname, params)
  do callStd;
  #);

glTexGenfv: glTexGendv(# #);

glTexGeniv: External
  (# coord: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *);
  enter (coord, pname, params)
  do callStd;
  #);

glGetTexGendv: External
  (# coord: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLdouble* *);
  enter (coord, pname, params)
  do callStd;
  #);

glGetTexGenfv: glGetTexGendv(# #);

glGetTexGeniv: External
  (# coord: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *);
  enter (coord, pname, params)
  do callStd;
  #);


glTexEnvf: External (* wrapped *)
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     param: @real (* GLfloat *);
  enter (target, pname, param)
     do 'wrapglTexEnvf'->CallC;
  #);
```

```
glTexEnvi: External
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     param: @int32 (* GLint *);
  enter (target, pname, param)
  do callStd;
  #);

glTexEnvfv: External
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *);
  enter (target, pname, params)
  do callStd;
  #);

glTexEnviv: External
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *);
  enter (target, pname, params)
  do callStd;
  #);

glGetTexEnvfv: External
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *);
  enter (target, pname, params)
  do callStd;
  #);

glGetTexEnviv: External
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *);
  enter (target, pname, params)
  do callStd;
  #);

glTexParameterf: External (* wrapped *)
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     param: @real (* GLfloat *);
  enter (target, pname, param)
  do 'wrapglTexParameterf'->CallC;
  #);

glTexParameteri: External
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     param: @int32 (* GLint *);
  enter (target, pname, param)
  do callStd;
  #);

glTexParameterfv: External
  (# target: @int32u (* GLenum *); pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *)
  enter (target, pname, params)
  do callStd;
  #);

glTexParameteriv: External
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
```

```
      params: @int32 (* GLint* *);
  enter (target, pname, params)
  do callStd;
  #);

glGetTexParameterfv: External
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *); params: @int32 (* GLfloat* *)
  enter (target, pname, params)
  do callStd;
  #);

glGetTexParameteriv: External
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *); params: @int32 (* GLint* *)
  enter (target, pname, params)
  do callStd;
  #);

glGetTexLevelParameterfv: External
  (# target: @int32u (* GLenum *); level: @int32 (* GLint *);
     pname: @int32u (* GLenum *); params: @int32 (* GLfloat* *)
  enter (target, level, pname, params)
  do callStd;
  #);

glGetTexLevelParameteriv: External
  (# target: @int32u (* GLenum *); level: @int32 (* GLint *);
     pname: @int32u (* GLenum *); params: @int32 (* GLint* *)
  enter (target, level, pname, params)
  do callStd;
  #);


glTexImage1D: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     internalFormat: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
     border: @int32 (* GLint *);
     format: @int32u (* GLenum *);
     type: @int32u (* GLenum *);
     pixels: @int32 (* GLvoid* *);
  enter (target, level, internalFormat, width, border, format, type, pixels)
  do callStd;
  #);

glTexImage2D: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     internalFormat: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
     height: @int32 (* GLsizei *);
     border: @int32 (* GLint *);
     format: @int32u (* GLenum *);
     type: @int32u (* GLenum *);
     pixels: @int32 (* GLvoid* *);
  enter (target, level, internalFormat, width, height, border, format, type, pixels)
  do callStd;
  #);

glGetTexImage: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     format: @int32u (* GLenum *);
     type: @int32u (* GLenum *);
```

```
      pixels: @int32 (* GLvoid* *);
  enter (target, level, format, type, pixels)
  do callStd;
  #);



(* 1.1 functions *)

glGenTextures: External
  (# n: @int32 (* GLsizei *);
     textures: @int32 (* GLuint* *);
  enter (n, textures)
  do callStd;
  #);

glDeleteTextures: External
  (# n: @int32 (* GLsizei *);
     textures: @int32 (* GLuint* *);
  enter (n, textures)
  do callStd;
  #);

glBindTexture: External
  (# target: @int32u (* GLenum *);
     texture: @int32 (* GLuint *);
  enter (target, texture)
  do callStd;
  #);

glPrioritizeTextures: External
  (# n: @int32 (* GLsizei *);
     textures: @int32 (* GLuint* *);
     priorities: @int32 (* GLclampf* *)
  enter (n, textures, priorities)
  do callStd;
  #);

glAreTexturesResident: External
  (# n: @int32 (* GLsizei *);
     textures: @int32 (* GLuint* *);
     residences: @int32 (* GLboolean* *);
     result: @int8u (* GLboolean *);
  enter (n, textures, residences)
  do callStd;
  exit result
  #);

glIsTexture: External
  (# texture: @int32 (* GLuint *);
     result: @int8u (* GLboolean *);
  enter texture
  do callStd;
  exit result
  #);


glTexSubImage1D: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     xoffset: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
     format: @int32u (* GLenum *);
     type: @int32u (* GLenum *);
     pixels: @int32 (* GLvoid* *);
  enter (target, level, xoffset, width, format, type, pixels)
```

```
  do callStd;
  #);


glTexSubImage2D: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     xoffset: @int32 (* GLint *);
     yoffset: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
     height: @int32 (* GLsizei *);
     format: @int32u (* GLenum *);
     type: @int32u (* GLenum *);
     pixels: @int32 (* GLvoid* *);
  enter (target, level, xoffset, yoffset, width, height, format, type, pixels)
  do callStd;
  #);


glCopyTexImage1D: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     internalformat: @int32u (* GLenum *);
     x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
     border: @int32 (* GLint *);
  enter (target, level, internalformat, x, y, width, border)
  do callStd;
  #);


glCopyTexImage2D: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     internalformat: @int32u (* GLenum *);
     x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
     height: @int32 (* GLsizei *);
     border: @int32 (* GLint *);
  enter (target, level, internalformat, x, y, width, height, border)
  do callStd;
  #);


glCopyTexSubImage1D: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     xoffset: @int32 (* GLint *);
     x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
  enter (target, level, xoffset, x, y, width)
  do callStd;
  #);


glCopyTexSubImage2D: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     xoffset: @int32 (* GLint *);
     yoffset: @int32 (* GLint *);
     x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
```

```
      height: @int32 (* GLsizei *);
  enter (target, level, xoffset, yoffset, x, y, width, height)
  do callStd;
  #);



(*
 * Evaluators
 *)

glMap1d: External
  (# target: @int32u (* GLenum *);
     u1: @real (* GLdouble *);
     u2: @real (* GLdouble *);
     stride: @int32 (* GLint *);
     order: @int32 (* GLint *);
     points: @int32 (* GLdouble* *);
  enter (target, u1, u2, stride, order, points)
  do callStd;
  #);

glMap1f: External
  (# target: @int32u (* GLenum *);
     u1: @real (* GLfloat *);
     u2: @real (* GLfloat *);
     stride: @int32 (* GLint *);
     order: @int32 (* GLint *);
     points: @int32 (* GLfloat* *);
  enter (target, u1, u2, stride, order, points)
  do callStd;
  #);

glMap2d: External
  (# target: @int32u (* GLenum *);
     u1: @real (* GLdouble *);
     u2: @real (* GLdouble *);
     ustride: @int32 (* GLint *);
     uorder: @int32 (* GLint *);
     v1: @real (* GLdouble *);
     v2: @real (* GLdouble *);
     vstride: @int32 (* GLint *);
     vorder: @int32 (* GLint *);
     points: @int32 (* GLdouble* *);
  enter (target, u1, u2, ustride, uorder, v1, v2, vstride, vorder, points)
  do callStd;
  #);

glMap2f: glMap2d(# #);

glGetMapdv: External
  (# target: @int32u (* GLenum *);
     query: @int32u (* GLenum *);
     v: @int32 (* GLdouble* *);
  enter (target, query, v)
  do callStd;
  #);

glGetMapfv: External
  (# target: @int32u (* GLenum *);
     query: @int32u (* GLenum *);
     v: @int32 (* GLfloat* *);
  enter (target, query, v)
  do callStd;
  #);
```

```
glGetMapiv: External
  (# target: @int32u (* GLenum *);
     query: @int32u (* GLenum *);
     v: @int32 (* GLint* *);
  enter (target, query, v)
  do callStd;
  #);

glEvalCoord1d: External
  (# u: @real (* GLdouble *);
  enter (u)
  do callStd;
  #);

glEvalCoord1f: glEvalCoord1d(# #);

glEvalCoord1dv: External
  (# u: @int32 (* GLdouble* *)
  enter (u)
  do callStd;
  #);

glEvalCoord1fv: glEvalCoord1dv(# #);

glEvalCoord2d: External
  (# u: @real (* GLdouble *);
     v: @real (* GLdouble *);
  enter (u, v)
  do callStd;
  #);

glEvalCoord2f: glEvalCoord2d(# #);

glEvalCoord2dv: External
  (# u: @int32 (* GLdouble* *)
  enter (u)
  do callStd;
  #);

glEvalCoord2fv: glEvalCoord2dv(# #);

glMapGrid1d: External
  (# un: @int32 (* GLint *);
     u1: @real (* GLdouble *);
     u2: @real (* GLdouble *);
  enter (un, u1, u2)
  do callStd;
  #);

glMapGrid1f: glMapGrid1d(# #);

glMapGrid2d: External
  (# un: @int32 (* GLint *);
     u1: @real (* GLdouble *);
     u2: @real (* GLdouble *);
     vn: @int32 (* GLint *);
     v1: @real (* GLdouble *);
     v2: @real (* GLdouble *);
  enter (un, u1, u2, vn, v1, v2)
  do callStd;
  #);

glMapGrid2f: glMapGrid2d(# #);

glEvalPoint1: External
```

```
  (# i: @int32 (* GLint *)
  enter (i)
  do callStd;
  #);

glEvalPoint2: External
  (# i: @int32 (* GLint *);
     j: @int32 (* GLint *);
  enter (i, j)
  do callStd;
  #);

glEvalMesh1: External
  (# mode: @int32u (* GLenum *);
     i1: @int32 (* GLint *);
     i2: @int32 (* GLint *);
  enter (mode, i1, i2)
  do callStd;
  #);

glEvalMesh2: External
  (# mode: @int32u (* GLenum *);
     i1: @int32 (* GLint *);
     i2: @int32 (* GLint *);
     j1: @int32 (* GLint *);
     j2: @int32 (* GLint *);
  enter (mode, i1, i2, j1, j2)
  do callStd;
  #);



(*
 * Fog
 *)

glFogf: External (* wrapped *)
  (# pname: @int32u (* GLenum *);
     param: @real (* GLfloat *);
  enter (pname, param)
  do 'wrapglFogf'->CallC;
  #);

glFogi: External
  (# pname: @int32u (* GLenum *);
     param: @int32 (* GLint *);
  enter (pname, param)
  do callStd;
  #);

glFogfv: External
  (# pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *);
  enter (pname, params)
  do callStd;
  #);

glFogiv: External
  (# pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *);
  enter (pname, params)
  do callStd;
  #);
```

```
(*
 * Selection and Feedback
 *)

glFeedbackBuffer: External
  (# size: @int32 (* GLsizei *);
     type: @int32u (* GLenum *);
     buffer: @int32 (* GLfloat* *);
  enter (size, type, buffer)
  do callStd;
  #);

glPassThrough: External (* wrapped *)
  (# token: @real (* GLfloat *);
  enter (token)
  do 'wrapglPassThrough'−>CallC;
  #);

glSelectBuffer: External
  (# size: @int32 (* GLsizei *);
     buffer: @int32 (* GLuint* *);
  enter (size, buffer)
  do callStd;
  #);

glInitNames: External
  (#
  do callStd;
  #);

glLoadName: External
  (# name: @int32 (* GLuint *);
  enter (name)
  do callStd;
  #);

glPushName: External
  (# name: @int32 (* GLuint *);
  enter (name)
  do callStd;
  #);

glPopName: External
  (#
  do callStd;
  #);



(*
 * 1.0 Extensions
 *)

(* GL_EXT_blend_minmax *)
glBlendEquationEXT: External
  (# mode: @int32u (* GLenum *);
  enter (mode)
  do callStd;
  #);

(* GL_EXT_blend_color *)
glBlendColorEXT: External
  (# red: @real (* GLclampf *);
     green: @real (* GLclampf *);
     blue: @real (* GLclampf *);
     alpha: @real (* GLclampf *);
```

```
  enter (red, green, blue, alpha)
  do callStd
  #);

(* GL_EXT_polygon_offset *)
(* Don't work *)
glPolygonOffsetEXT: External (* Wrapped *)
  (# factor: @real (* GLfloat *);
     bias: @real (* GLfloat *);
  enter (factor, bias)
  do 'wrapglPolygonOffsetEXT'−>CallC;
  #);



(* GL_EXT_vertex_array *)

glVertexPointerEXT: External
  (# size: @int32 (* GLint *);
     type: @int32u (* GLenum *);
     stride: @int32 (* GLsizei *);
     count: @int32 (* GLsizei *);
     ptr: @int32 (* GLvoid* *);
  enter (size, type, stride, count, ptr)
  do callStd;
  #);

glNormalPointerEXT: External
  (# type: @int32u (* GLenum *);
     stride: @int32 (* GLsizei *);
     count: @int32 (* GLsizei *);
     ptr: @int32 (* GLvoid* *);
  enter (type, stride, count, ptr)
  do callStd;
  #);

glColorPointerEXT: External
  (# size: @int32 (* GLint *);
     type: @int32u (* GLenum *);
     stride: @int32 (* GLsizei *);
     count: @int32 (* GLsizei *);
     ptr: @int32 (* GLvoid* *);
  enter (size, type, stride, count, ptr)
  do callStd;
  #);

glIndexPointerEXT: External
  (# type: @int32u (* GLenum *);
     stride: @int32 (* GLsizei *);
     count: @int32 (* GLsizei *);
     ptr: @int32 (* GLvoid* *);
  enter (type, stride, count, ptr)
  do callStd;
  #);

glTexCoordPointerEXT: External
  (# size: @int32 (* GLint *);
     type: @int32u (* GLenum *);
     stride: @int32 (* GLsizei *);
     count: @int32 (* GLsizei *);
     ptr: @int32 (* GLvoid* *);
  enter (size, type, stride, count, ptr)
  do callStd;
  #);

glEdgeFlagPointerEXT: External
```

```
  (# stride: @int32 (* GLsizei *);
     count: @int32 (* GLsizei *);
     ptr: @int32 (* GLboolean* *);
  enter (stride, count, ptr)
  do callStd;
  #);

glGetPointervEXT: External
  (# pname: @int32u (* GLenum *);
     params: @int32 (* void** *);
  enter (pname, params)
  do callStd;
  #);

glArrayElementEXT: External
  (# i: @int32 (* GLint *);
  enter (i)
  do callStd;
  #);

glDrawArraysEXT: External
  (# mode: @int32u (* GLenum *);
     first: @int32 (* GLint *);
     count: @int32 (* GLsizei *);
  enter (mode, first, count)
  do callStd;
  #);



(* GL_EXT_texture_object *)

glGenTexturesEXT: External
  (# n: @int32 (* GLsizei *);
     textures: @int32 (* GLuint* *);
  enter (n, textures)
  do callStd;
  #);

glDeleteTexturesEXT: External
  (# n: @int32 (* GLsizei *);
     textures: @int32 (* GLuint* *);
  enter (n, textures)
  do callStd;
  #);

glBindTextureEXT: External
  (# target: @int32u (* GLenum *);
     texture: @int32 (* GLuint *);
  enter (target, texture)
  do callStd;
  #);

glPrioritizeTexturesEXT: External
  (# n: @int32 (* GLsizei *);
     textures: @int32 (* GLuint* *);
     priorities: @int32 (* GLclampf* *);
  enter (n, priorities)
  do callStd;
  #);

glAreTexturesResidentEXT: External
  (#  n: @int32 (* GLsizei *);
     textures: @int32 (* GLuint* *);
     residences: @int32 (* GLboolean* *);
     result: @int8u (* GLboolean *)
```

```
  enter (n, textures, residences)
  do callStd;
  exit result
  #);

glIsTextureEXT: External
  (# texture: @int32 (* GLuint *);
     result: @int8u (* GLboolean *);
  enter (texture)
  do callStd;
  exit (result)
  #);


(* GL_EXT_texture3D *)

glTexImage3DEXT: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     internalFormat: @int32u (* GLenum *);
     width: @int32 (* GLsizei *);
     height: @int32 (* GLsizei *);
     depth: @int32 (* GLsizei *);
     border: @int32 (* GLint *);
     format: @int32u (* GLenum *);
     type: @int32u (* GLenum *);
     pixels: @int32 (* GLvoid* *);
  enter (target, level, internalFormat, width, height, depth, border, format, type, pixels)
  do callStd;
  #);

glTexSubImage3DEXT: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     xoffset: @int32 (* GLint *);
     yoffset: @int32 (* GLint *);
     zoffset: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
     height: @int32 (* GLsizei *);
     depth: @int32 (* GLsizei *);
     format: @int32u (* GLenum *);
     type: @int32u (* GLenum *);
     pixels: @int32 (* GLvoid* *);
  enter (target, level, xoffset, yoffset, zoffset, width, height, depth, format, type, pixels)
  do callStd;
  #);

glCopyTexSubImage3DEXT: External
  (# target: @int32u (* GLenum *);
     level: @int32 (* GLint *);
     xoffset: @int32 (* GLint *);
     yoffset: @int32 (* GLint *);
     zoffset: @int32 (* GLint *);
     x: @int32 (* GLint *);
     y: @int32 (* GLint *);
     width: @int32 (* GLsizei *);
     height: @int32 (* GLsizei *);
  enter (target, level, xoffset, yoffset, zoffset, x, y, width, height)
  do callStd;
  #);



(* EXT_color_table *)

glColorTableEXT: External
```

```
  (# target: @int32u (* GLenum *);
     internalformat: @int32u (* GLenum *);
     width: @int32 (* GLsizei *);
     format: @int32u (* GLenum *);
     type: @int32u (* GLenum *);
     table: @int32 (* GLvoid* *);
  enter (target, internalformat, width, format, type, table)
  do callStd;
  #);

glColorSubTableEXT: External
  (# target: @int32u (* GLenum *);
     start: @int32 (* GLsizei *);
     count: @int32 (* GLsizei *);
     format: @int32u (* GLenum *);
     type: @int32u (* GLenum *);
     data: @int32 (* GLvoid* *);
  enter (target, start, count, format, type, data)
  do callStd;
  #);

glGetColorTableEXT: External
  (# target: @int32u (* GLenum *);
     format: @int32u (* GLenum *);
     type: @int32u (* GLenum *);
     table: @int32 (* GLvoid* *);
  enter (target, format, type, table)
  do callStd;
  #);

glGetColorTableParameterfvEXT: External
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *);
  enter (target, pname, params)
  do callStd;
  #);

glGetColorTableParameterivEXT: External
  (# target: @int32u (* GLenum *);
     pname: @int32u (* GLenum *);
     params: @int32 (* GLint* *)
  enter (target, pname, params)
  do callStd;
  #);



(* GL_EXT_point_parameters *)

glPointParameterfEXT: External (* wrapped *)
  (# pname: @int32u (* GLenum *);
     param: @real (* GLfloat *);
  enter (pname, param)
  do 'wrapglPointParameterfEXT'->CallC;
  #);

glPointParameterfvEXT: External
  (# pname: @int32u (* GLenum *);
     params: @int32 (* GLfloat* *);
  enter (pname, params)
  do callStd;
  #)
```

23.2 Glb Interface

```
ORIGIN 'opengl'
-- lib:attributes --

(* small patterns, for pretty use of repetitions as vectors *)
Vector3d:
  (# data:[3]@real;
     copy:(# exit (data[1],data[2],data[3]) #);
  enter (data[1],data[2],data[3])
  exit @@data[1]
  #);

Vector4d:
  (# data:[4]@real;
     copy:(# exit (data[1],data[2],data[3],data[4]) #);
  enter (data[1],data[2],data[3],data[4])
  exit @@data[1]
  #);

Vector3i:
  (# data:[4]@integer;
     copy:(# exit (data[1],data[2],data[3],data[4]) #);
  enter (data[1],data[2],data[3],data[4])
  exit @@data[1]
  #);

Vector4i:
  (# data:[4]@integer;
     copy:(# exit (data[1],data[2],data[3],data[4]) #);
  enter (data[1],data[2],data[3],data[4])
  exit @@data[1]
  #);



glBWireCube: (* Not very usefull, does not generate any normals/texture coords *)
  (# size:@real;
  enter size
  do GL_LINE_LOOP->glBegin;
     (0,0,0)->glVertex3i;
     (size,0,0)->glVertex3d;
     (size,0,-size)->glVertex3d;
     (0,0,-size)->glVertex3d;
     glEnd;
     GL_LINE_LOOP->glBegin;
     (0,size,0)->glVertex3d;
     (size,size,0)->glVertex3d;
     (size,size,-size)->glVertex3d;
     (0,size,-size)->glVertex3d;
     glEnd;
     GL_LINES->glBegin;
     (0,0,0)->glVertex3i;
     (0,size,0)->glVertex3d;
     (size,0,0)->glVertex3d;
     (size,size,0)->glVertex3d;
     (size,0,-size)->glVertex3d;
     (size,size,-size)->glVertex3d;
     (0,0,-size)->glVertex3d;
     (0,size,-size)->glVertex3d;
     glEnd;
  #);

glBSolidCube: (*NB does not compute vertex normals *)
```

```
(# size:@real;
enter size
do GL_POLYGON->glBegin; (* bottom *)
   (0,0,0)->glVertex3i;
   (0,0,size)->glVertex3d;
   (size,0,size)->glVertex3d;
   (size,0,0)->glVertex3d;
   glEnd;
   GL_POLYGON->glBegin; (* top *)
   (0,size,0)->glVertex3d;
   (0,size,size)->glVertex3d;
   (size,size,size)->glVertex3d;
   (size,size,0)->glVertex3d;
   glEnd;
   GL_POLYGON->glBegin; (* front *)
   (0,0,size)->glVertex3d;
   (size,0,size)->glVertex3d;
   (size,size,size)->glVertex3d;
   (0,size,size)->glVertex3d;
   glEnd;
   GL_POLYGON->glBegin; (* back *)
   (0,0,0)->glVertex3d;
   (size,0,0)->glVertex3d;
   (size,size,0)->glVertex3d;
   (0,size,0)->glVertex3d;
   glEnd;
   GL_POLYGON->glBegin; (* left *)
   (0,0,0)->glVertex3i;
   (0,0,size)->glVertex3d;
   (0,size,size)->glVertex3d;
   (0,size,0)->glVertex3d;
   glEnd;
   GL_POLYGON->glBegin; (* right *)
   (size,0,0)->glVertex3d;
   (size,0,size)->glVertex3d;
   (size,size,size)->glVertex3d;
   (size,size,0)->glVertex3d;
   glEnd;
#)
```

23.3 Glu Interface

```
ORIGIN 'opengl';

-- lib: attributes --

(* GLU constants *)

(* GLU Normal vectors *)
GLU_SMOOTH: (# exit 100000 #);
GLU_FLAT: (# exit 100001 #);
GLU_NONE: (# exit 100002 #);

(* GLU Quadric draw styles *)
GLU_POINT: (# exit 100010 #);
GLU_LINE: (# exit 100011 #);
GLU_FILL: (# exit 100012 #);
GLU_SILHOUETTE: (# exit 100013 #);

(* GLU Quadric orientation *)
GLU_OUTSIDE: (# exit 100020 #);
GLU_INSIDE: (# exit 100021 #);

(* GLU Tesselator *)
GLU_BEGIN: (# exit 100100 #);
GLU_VERTEX: (# exit 100101 #);
GLU_END: (# exit 100102 #);
GLU_ERROR: (# exit 100103 #);
GLU_EDGE_FLAG: (# exit 100104 #);

(* GLU Contour types *)
GLU_CW: (# exit 100120 #);
GLU_CCW: (# exit 100121 #);
GLU_INTERIOR: (# exit 100122 #);
GLU_EXTERIOR: (# exit 100123 #);
GLU_UNKNOWN: (# exit 100124 #);

(* GLU Tesselation errors *)
GLU_TESS_ERROR1: (# exit 100151 #);
GLU_TESS_ERROR2: (# exit 100152 #);
GLU_TESS_ERROR3: (# exit 100153 #);
GLU_TESS_ERROR4: (# exit 100154 #);
GLU_TESS_ERROR5: (# exit 100155 #);
GLU_TESS_ERROR6: (# exit 100156 #);
GLU_TESS_ERROR7: (# exit 100157 #);
GLU_TESS_ERROR8: (# exit 100158 #);
GLU_TESS_ERROR9: (# exit 100159 #);

(* NURBS *)
GLU_AUTO_LOAD_MATRIX: (# exit 100200 #);
GLU_CULLING: (# exit 100201 #);
GLU_PARAMETRIC_TOLERANC: (# exit 100202 #);
GLU_SAMPLING_TOLERANCE: (# exit 100203 #);
GLU_DISPLAY_MODE: (# exit 100204 #);
GLU_SAMPLING_METHOD: (# exit 100205 #);
GLU_U_STEP: (# exit 100206 #);
GLU_V_STEP: (# exit 100207 #);
GLU_PATH_LENGTH: (# exit 100215 #);
GLU_PARAMETRIC_ERROR: (# exit 100216 #);
GLU_DOMAIN_DISTANCE: (# exit 100217 #);

GLU_MAP1_TRIM_2: (# exit 100210 #);
GLU_MAP1_TRIM_3: (# exit 100211 #);
```

```
GLU_OUTLINE_POLYGON: (# exit 100240 #);
GLU_OUTLINE_PATCH: (# exit 100241 #);

GLU_NURBS_ERROR1: (# exit  100251 #);
GLU_NURBS_ERROR2: (# exit  100252 #);
GLU_NURBS_ERROR3: (# exit  100253 #);
GLU_NURBS_ERROR4: (# exit  100254 #);
GLU_NURBS_ERROR5: (# exit  100255 #);
GLU_NURBS_ERROR6: (# exit  100256 #);
GLU_NURBS_ERROR7: (# exit  100257 #);
GLU_NURBS_ERROR8: (# exit  100258 #);
GLU_NURBS_ERROR9: (# exit  100259 #);
GLU_NURBS_ERROR10: (# exit 100260 #);
GLU_NURBS_ERROR11: (# exit 100261 #);
GLU_NURBS_ERROR12: (# exit 100262 #);
GLU_NURBS_ERROR13: (# exit 100263 #);
GLU_NURBS_ERROR14: (# exit 100264 #);
GLU_NURBS_ERROR15: (# exit 100265 #);
GLU_NURBS_ERROR16: (# exit 100266 #);
GLU_NURBS_ERROR17: (# exit 100267 #);
GLU_NURBS_ERROR18: (# exit 100268 #);
GLU_NURBS_ERROR19: (# exit 100269 #);
GLU_NURBS_ERROR20: (# exit 100270 #);
GLU_NURBS_ERROR21: (# exit 100271 #);
GLU_NURBS_ERROR22: (# exit 100272 #);
GLU_NURBS_ERROR23: (# exit 100273 #);
GLU_NURBS_ERROR24: (# exit 100274 #);
GLU_NURBS_ERROR25: (# exit 100275 #);
GLU_NURBS_ERROR26: (# exit 100276 #);
GLU_NURBS_ERROR27: (# exit 100277 #);
GLU_NURBS_ERROR28: (# exit 100278 #);
GLU_NURBS_ERROR29: (# exit 100279 #);
GLU_NURBS_ERROR30: (# exit 100280 #);
GLU_NURBS_ERROR31: (# exit 100281 #);
GLU_NURBS_ERROR32: (# exit 100282 #);
GLU_NURBS_ERROR33: (# exit 100283 #);
GLU_NURBS_ERROR34: (# exit 100284 #);
GLU_NURBS_ERROR35: (# exit 100285 #);
GLU_NURBS_ERROR36: (# exit 100286 #);
GLU_NURBS_ERROR37: (# exit 100287 #);

(* GLU Errors *)
GLU_INVALID_ENUM: (# exit 100900 #);
GLU_INVALID_VALUE: (# exit 10090 #);
GLU_OUT_OF_MEMORY: (# exit 10090 #);
GLU_INCOMPATIBLE_GL_VERSION: (# exit 100903 #);

(* GLU 1.1 *)
GLU_VERSION: (# exit 100800 #);
GLU_EXTENSIONS: (# exit 100801 #);

(* Miscellaneous GLU functions *)

gluLookAt: External
  (# eyex, eyey, eyez: @real;
     centerx, centery, centerz: @real;
     upx, upy, upz: @real;
  enter (eyex,eyey,eyez,centerx,centery,centerz,upx,upy,upz)
  do callStd;
  #);

gluOrtho2D: External
  (# left,right,bottom,top: @real;
  enter (left,right,bottom,top)
  do callStd;
  #);
```

```
gluPerspective: External
  (# fovy,aspect,zNear,zFar: @real;
  enter(fovy,aspect,zNear,zFar)
  do callStd;
  #);

gluPickMatrix: External
  (# x,y,width,height: @real;
     viewport: @integer;
  enter (x,y,width,height,viewport)
  do callStd;
  #);

gluProject: (* Wrapped ,not done*)
  (# objx,objy,objz: @real;
     modelMatrix,projMatrix,viewport: @integer;
     winx,winy,winz: @integer;
     result: @integer;
  enter (objx,objy,objz,modelMatrix,projMatrix,viewport,winx,winy,winz)
  exit result
  #);

gluUnProject: (* Wrapped ,not done*)
  (# winx,winy,winz: @real;
     modelMatrix,projMatrix,viewport: @integer;
     objx,objy,objz: @integer;
     result: @integer;
  enter (winx,winy,winz,modelMatrix,projMatrix,viewport,objx,objy,objz)
  exit result
  #);

gluErrorString: External
  (# errorCode: @integer;
     result: [0]@char;
  enter errorCode
  do CallStd;
  exit result
  #);


(* GLU Mipmapping and image scaling *)

gluScaleImage: External
  (# format: @integer;
     widthin,heightin: @integer;
     typein,datain: @integer;
     widthout,heightout: @integer;
     typeout,dataout: @integer;
     result: @integer;
  enter (format,widthin,heightin,typein,datain,widthout,heightout,typeout,dataout)
  do CallStd;
  exit result
  #);

gluBuild1DMipmaps: External
  (# target,components,width,format: @integer;
     type,data: @integer;
     result: @integer;
  enter (target,components,width,format,type,data)
  do CallStd;
  exit result
  #);

gluBuild2DMipmaps: External
  (# target,components,width,format: @integer;
```

```
      type,data: @integer;
      result: @integer;
  enter (target,components,width,format,type,data)
  do CallStd;
  exit result
  #);


(* GLU Quadrics *)

gluNewQuadric: External
  (# quadobj: @integer;
  do CallStd;
  exit quadobj
  #);

gluDeleteQuadric: External
  (# obj: @integer
  enter obj
  do CallStd;
  #);

gluQuadricDrawStyle: External
  (# quadObject: @integer;
     drawStyle: @integer;
  enter (quadObject,drawStyle)
  do CallStd;
  #);

gluQuadricOrientation: External
  (# quadObject: @integer;
     orientation: @integer;
  enter (quadObject,orientation)
  do CallStd;
  #);

gluQuadricNormals: External
  (# quadObject: @integer;
     normals: @integer;
  enter (quadObject,normals)
  do CallStd;
  #);

gluQuadricTexture: External
  (# quadObject: @integer;
     textureCoords: @char; (* GLboolean *)
  enter (quadObject,textureCoords)
  do CallStd;
  #);

(*
gluQuadricCallback: External
  (# qobj: @integer;
     which: @integer;
     void ( *fn)()
  #);
 *)

gluCylinder: External
  (# qobj: @integer;
     baseRadius, topRadius, height: @real;
     slices, stacks: @integer;
  enter (qobj,baseRadius,topRadius,height,slices,stacks)
  do callStd;
  #);
```

```
gluSphere: External
  (# qobj: @integer;
     radius: @real;
     slices,stacks: @integer;
  enter (qobj,radius,slices,stacks)
  do CallStd;
  #);

gluDisk: External
  (# qobj: @integer;
     innerRadius,outerRadius: @real;
     slices, loops: @integer;
  enter (qobj,innerRadius,outerRadius,slices,loops)
  do CallStd;
  #);

gluPartialDisk: External
  (# qobj: @integer;
     innerRadius, outerRadius: @real;
     slices, loops: @integer;
     startAngle, sweepAngle: @real;
  enter (qobj,innerRadius,outerRadius,slices,loops,startAngle,sweepAngle)
  do callStd;
  #);

(* GLU Nurbs *)

gluNewNurbsRenderer: External
  (# nurbsobj: @integer;
  do CallStd;
  exit nurbsobj
  #);

gluDeleteNurbsRenderer: External
  (# nurbsobj: @integer;
  enter nurbsobj
  do CallStd;
  #);

(*
gluLoadSamplingMatrices: External
  (# nobj: @integer;
     GLfloat modelMatrix[16],
     GLfloat projMatrix[16],
     GLint viewport[4]
  #);
 *)

gluNurbsProperty: External (* wrapped *)
  (# nobj: @integer;
     property: @integer;
     value: @real;
  enter (nobj,property,value)
  do 'wrapgluNurbsProperty'->CallC;
  #);

gluGetNurbsProperty: External
  (# nobj,property: @integer;
     value: @integer; (* GLfloat* *)
  enter (nobj,property,value)
  do CallStd;
  #);

gluBeginCurve: External
  (# nobj: @integer;
  enter nobj
```

```
  do CallStd;
  #);

gluEndCurve: External
  (# nobj: @integer;
  enter nobj
  do CallStd;
  #);

gluNurbsCurve: External (* wrapped, not done *)
  (# nobj, nknots: @integer;
     knot: @integer; (* GLfloat* *)
     stride: @integer;
     ctlarray: @integer; (* GLfloat* *)
     order, type: @integer;
  enter (nobj,nknots,knot,stride,ctlarray,order,type)
  #);

gluBeginSurface: External
  (# nobj: @integer;
  enter nobj
  do CallStd;
  #);

gluEndSurface: External
  (# nobj: @integer;
  enter nobj
  do CallStd;
  #);

gluNurbsSurface: External (* wrapped, not done *)
  (# nobj: @integer;
     sknot_count: @integer;
     sknot: @integer; (* GLfloat* *)
     tknot_count: @integer;
     tknot: @integer; (* GLfloat* *)
     s_stride, t_stride: @integer;
     ctlarray: @integer; (* GLfloat* *)
     sorder, torder,type: @integer;
  enter (nobj,sknot_count,sknot,tknot_count,tknot,s_stride,t_stride,ctlarray,sorder,torder,type
  #);

gluBeginTrim: External
  (# nobj: @integer;
  enter nobj
  do CallStd;
  #);

gluEndTrim: External
  (# nobj: @integer;
  enter nobj
  do CallStd;
  #);

gluPwlCurve: External (* wrapped, not done *)
  (# nobj, count: @integer;
     array: @integer; (* GLfloat* *)
     stride, type: @integer;
  enter (nobj,count,array,stride,type)
  #);


(* GLU Polygon tesselation *)

gluNewTess: External
  (# triangulatorObj: @integer;
```

```
  do CallStd;
  exit triangulatorObj
  #);

(*
 gluTessCallback
 *)

gluDeleteTess: External
  (# tobj: @integer;
  enter tobj
  do CallStd;
  #);

gluBeginPolygon: External
  (#
  tobj: @integer;
  enter tobj
  do CallStd;
  #);

gluEndPolygon: External
  (# tobj: @integer;
  enter tobj
  do CallStd;
  #);

gluNextContour: External
  (# tobj,type: @integer;
  enter (tobj,type)
  do CallStd;
  #);

gluTessVertex: External
  (# tobj: @integer;
     v: @integer; (* GLdouble* *)
     data: @integer; (* void* *)
  enter (tobj,v,data)
  do CallStd;
  #);


(* GLU 1.1 *)

gluGetString: External
  (# name: @integer;
     result: [0]@char;
  enter name
  do CallStd;
  exit result
  #)
```

23.4 Wgl Interface

```
ORIGIN 'opengl';

--lib:attributes--

(*------ GLW ---------- *)
(* note this is not complete,
 * only intendet for internal use,
 * as programs is not platform independent.
 *
 *)

(* Different constants *)

(* pixel types *)
PFD_TYPE_RGBA:(# exit 0 #);
PFD_TYPE_COLORINDEX:(# exit 1 #);

(* layer types *)
PFD_MAIN_PLANE:(# exit 0 #);
PFD_OVERLAY_PLANE:(# exit 1 #);
PFD_UNDERLAY_PLANE:(# exit (-1) #);

(* PIXELFORMATDESCRIPTOR flags *)
PFD_DOUBLEBUFFER:(# exit 0x00000001 #);
PFD_STEREO:(# exit 0x00000002 #);
PFD_DRAW_TO_WINDOW:(# exit 0x00000004 #);
PFD_DRAW_TO_BITMAP:(# exit 0x00000008 #);
PFD_SUPPORT_GDI:(# exit 0x00000010 #);
PFD_SUPPORT_OPENGL:(# exit 0x00000020 #);
PFD_GENERIC_FORMAT:(# exit 0x00000040 #);
PFD_NEED_PALETTE:(# exit 0x00000080 #);
PFD_NEED_SYSTEM_PALETTE:(# exit 0x00000100 #);
PFD_SWAP_EXCHANGE:(# exit 0x00000200 #);
PFD_SWAP_COPY:(# exit 0x00000400 #);
PFD_SWAP_LAYER_BUFFERS:(# exit 0x00000800 #);
PFD_GENERIC_ACCELERATED:(# exit 0x00001000 #);

(* PIXELFORMATDESCRIPTOR flags for use in ChoosePixelFormat only *)
PFD_DEPTH_DONTCARE:(# exit 0x20000000 #);
PFD_DOUBLEBUFFER_DONTCARE:(# exit 0x40000000 #);
PFD_STEREO_DONTCARE:(# exit 0x80000000 #);

SwapBuffers:external
  (# hdc:@integer; (* HDC* *)
     result:@integer;
  enter hdc
  do CallStd
  exit result
  #);

(* LAYERPLANEDESCRIPTOR flags *)
LPD_DOUBLEBUFFER: (# exit 0x00000001 #);
LPD_STEREO: (# exit 0x00000002 #);
LPD_SUPPORT_GDI: (# exit 0x00000010 #);
LPD_SUPPORT_OPENGL: (# exit 0x00000020 #);
LPD_SHARE_DEPTH: (# exit 0x00000040 #);
LPD_SHARE_STENCIL: (# exit 0x00000080 #);
LPD_SHARE_ACCUM: (# exit 0x00000100 #);
LPD_SWAP_EXCHANGE: (# exit 0x00000200 #);
LPD_SWAP_COPY: (# exit 0x00000400 #);
LPD_TRANSPARENT: (# exit 0x00001000 #);
```

```
LPD_TYPE_RGBA: (# exit 0 #);
LPD_TYPE_COLORINDEX: (# exit 1 #);

(* wglSwapLayerBuffers flags *)
WGL_SWAP_MAIN_PLANE: (# exit 0x00000001 #);
WGL_SWAP_OVERLAY1: (# exit 0x00000002 #);
WGL_SWAP_OVERLAY2: (# exit 0x00000004 #);
WGL_SWAP_OVERLAY3: (# exit 0x00000008 #);
WGL_SWAP_OVERLAY4: (# exit 0x00000010 #);
WGL_SWAP_OVERLAY5: (# exit 0x00000020 #);
WGL_SWAP_OVERLAY6: (# exit 0x00000040 #);
WGL_SWAP_OVERLAY7: (# exit 0x00000080 #);
WGL_SWAP_OVERLAY8: (# exit 0x00000100 #);
WGL_SWAP_OVERLAY9: (# exit 0x00000200 #);
WGL_SWAP_OVERLAY10: (# exit 0x00000400 #);
WGL_SWAP_OVERLAY11: (# exit 0x00000800 #);
WGL_SWAP_OVERLAY12: (# exit 0x00001000 #);
WGL_SWAP_OVERLAY13: (# exit 0x00002000 #);
WGL_SWAP_OVERLAY14: (# exit 0x00004000 #);
WGL_SWAP_OVERLAY15: (# exit 0x00008000 #);
WGL_SWAP_UNDERLAY1: (# exit 0x00010000 #);
WGL_SWAP_UNDERLAY2: (# exit 0x00020000 #);
WGL_SWAP_UNDERLAY3: (# exit 0x00040000 #);
WGL_SWAP_UNDERLAY4: (# exit 0x00080000 #);
WGL_SWAP_UNDERLAY5: (# exit 0x00100000 #);
WGL_SWAP_UNDERLAY6: (# exit 0x00200000 #);
WGL_SWAP_UNDERLAY7: (# exit 0x00400000 #);
WGL_SWAP_UNDERLAY8: (# exit 0x00800000 #);
WGL_SWAP_UNDERLAY9: (# exit 0x01000000 #);
WGL_SWAP_UNDERLAY10: (# exit 0x02000000 #);
WGL_SWAP_UNDERLAY11: (# exit 0x04000000 #);
WGL_SWAP_UNDERLAY12: (# exit 0x08000000 #);
WGL_SWAP_UNDERLAY13: (# exit 0x10000000 #);
WGL_SWAP_UNDERLAY14: (# exit 0x20000000 #);
WGL_SWAP_UNDERLAY15: (# exit 0x40000000 #);

wglDescribeLayerPlane: external
  (# hdc, iPixelFormat, iLayerFormat, nbytes, plpd :@integer (* HDC, int, int, UINT, LPLAYERPLA
     result:@integer;
  enter (hdc, iPixelFormat, iLayerFormat, nbytes, plpd)
  do CallStd
  exit result
  #);
wglSetLayerPaletteEntries: external
  (# hdc,iLayerPlane,iStart,cEntries,pcr :@integer; (* HDC, int, int, int, CONST COLORREF *)
     result:@integer;
  enter (hdc,iLayerPlane,iStart,cEntries,pcr)
  do CallStd
  exit result
  #);
wglGetLayerPaletteEntries: external
  (# hdc,iLayerPlane,iStart,cEntries,pcr :@integer; (* HDC, int, int, int, COLORREF* *)
     result:@integer;
  enter (hdc,iLayerPlane,iStart,cEntries,pcr)
  do CallStd
  exit result
  #);
wglRealizeLayerPalette: external
  (# hdc,iLayerPlane,bRealize :@integer; (* HDC, int, BOOL *);
     result:@integer;
  enter (hdc,iLayerPlane,bRealize)
  do CallStd
  exit result
  #);

wglSwapLayerBuffers: external
```

```
  (# hdc,fuPlanes :@integer; (* HDC*, uint *)
     result:@integer;
  enter (hdc,fuPlanes)
  do CallStd
  exit result
  #);

(* utility functions in external/betaopengl.c *)
setPixelFormat:external
  (# hdc:@integer; (* HDC* *)
     result:@integer;
  enter hdc
  exit result
  #);
setPixelFormatOverlay:external
  (# hdc, type, flags, NumPaletteEntries, Palette:@integer; (* HDC*, BYTE, DWORD *)
     result:@integer;
  enter (hdc, type, flags, NumPaletteEntries, Palette)
  exit result
  #);

wglCopyContext:external
  (# source,dest,mask:@integer; (* HGLRC, HGLRC, UINT *)
  enter (source,dest,mask)
  #);


wglCreateContext:external
  (# hdc:@integer; (* HDC *)
     glrc:@integer; (* HGLRC *)
  enter hdc
  do CallStd
  exit glrc
  #);

wglCreateLayerContext:external
  (# hdc,iLayerPlane:@integer; (* HDC, int *)
     glrc:@integer; (* HGLRC *)
  enter (hdc,iLayerPlane)
  do CallStd
  exit glrc
  #);

wglDeleteContext:external
  (# hglrc:@integer; (*HGLRC *)
     result:@integer;
  enter hglrc
  do CallStd;
  exit result
  #);

wglGetCurrentContext:external
  (# hglrc:@integer;
  do CallStd;
  exit hglrc
  #);

wglGetCurrentDC:external
  (# hdc:@integer;
  do CallStd;
  exit hdc
  #);

(*
 * Do we need this ?
 *
```

```
 * wglGetProcAddress:external
 (# (LPCSTR);#);
 *)


wglMakeCurrent:external
  (# hdc,hglrc:@integer;(*HDC, HGLRC*)
     result:@integer;
  enter (hdc,hglrc)
  do CallStd;
  exit result
  #);

wglShareLists:external
  (# hglrc1,hglrc2:@integer; (*HGLRC, HGLRC*)
     result:@integer;
  enter (hglrc1,hglrc2)
  do CallStd;
  exit result
  #)
```

23.5 Glx Interface

```
ORIGIN 'opengl';

--lib:attributes--

(*------ GLX ---------- *)
(* note this is not complete,
 * only intendet for internal use,
 * as programs is not platform independent
 *)

(* --------GLX_CONFIGURATION-------- 83 *)
GLX_USE_GL:(# exit 1 #);
GLX_BUFFER_SIZE:(# exit 2 #);
GLX_LEVEL:(# exit 3 #);
GLX_RGBA:(# exit 4 #);
GLX_DOUBLEBUFFER:(# exit 5 #);
GLX_STEREO:(# exit 6 #);
GLX_AUX_BUFFERS:(# exit 7 #);
GLX_RED_SIZE:(# exit 8 #);
GLX_GREEN_SIZE:(# exit 9 #);
GLX_BLUE_SIZE:(# exit 10 #);
GLX_ALPHA_SIZE:(# exit 11 #);
GLX_DEPTH_SIZE:(# exit 12 #);
GLX_STENCIL_SIZE:(# exit 13 #);
GLX_ACCUM_RED_SIZE:(# exit 14 #);
GLX_ACCUM_GREEN_SIZE:(# exit 15 #);
GLX_ACCUM_BLUE_SIZE:(# exit 16 #);
GLX_ACCUM_ALPHA_SIZE:(# exit 17 #);

(* GLX_EXT_visual_info extension *)
GLX_X_VISUAL_TYPE_EXT:(# exit 0x22 #);
GLX_TRANSPARENT_TYPE_EXT:(# exit 0x23 #);
GLX_TRANSPARENT_INDEX_VALUE_EXT:(# exit 0x24 #);
GLX_TRANSPARENT_RED_VALUE_EXT:(# exit 0x25 #);
GLX_TRANSPARENT_GREEN_VALUE_EXT:(# exit 0x26 #);
GLX_TRANSPARENT_BLUE_VALUE_EXT:(# exit 0x27 #);
GLX_TRANSPARENT_ALPHA_VALUE_EXT:(# exit 0x2 #);


(*---------Various functions ---------------*)

glXChooseVisual:external (* 171 *)
  (# display:@integer;
     screen:@integer;
     attributes:@integer;
     visual:@integer;
  enter (display,screen,attributes)
  exit visual
  #);

glXCopyContext:external
  (# display:@integer;
     src,dst,mask:@integer;
  enter (display,src,dst,mask)
  #);

glXCreateContext:external (* 174 *)
  (# display:@integer;
     visual:@integer; (* found with glXChooseVisual *)
     sharelist:@integer; (* glXcontext *)
     direct:@integer;  (* 1 true, 0 false *)
     context:@integer;
```

```
  enter (display,visual,sharelist,direct)
  exit context
  #);

glXCreateGLXPixmap:external
  (# display:@integer;
     visual:@integer;
     pixmap, glxpixmap: @integer
  enter (display,visual,pixmap)
  exit glxpixmap
  #);

glXDestroyContext:external
  (# display:@integer;
     context:@integer;
  enter (display,context)
  #);

glXDestroyGLXPixmap:external
  (# display:@integer;
     glxpixmap:@integer;
  enter (display,glxpixmap)
  #);

glXGetClientString:external
  (# display:@integer;
     name:@integer;
     string: [0]@char;
  enter (display,name)
  exit string
  #);

glXGetConfig:external
  (# display:@integer;
     visual:@integer;
     attrib:@integer;
     value_ptr:@integer;
     status:@integer;
  enter (display,visual,attrib,value_ptr)
  exit status
  #);

glXGetCurrentDisplay:external
  (# display:@integer;
  exit display
  #);

glXGetCurrentDrawable:external
  (# drawable:@integer;
  exit drawable
  #);

glXIsDirect:external
  (# display,context:@integer;
     isdirect:@integer;
  enter (display,context)
  exit isdirect
  #);

glXMakeCurrent:external (* 179 *)
  (# display:@integer;
     drawable:@integer;
     context:@integer;
     result:@integer;
  enter(display,drawable,context)
  exit result
```

```
  #);

glXQueryExtension:external
  (# display,errorBase_ptr,eventBase_ptr:@integer;
     has_extension:@boolean
  enter (display,errorBase_ptr,eventBase_ptr)
  exit has_extension
  #);

glXQueryExtensionString:external
  (# display,screen:@integer;
     string: [0]@char
  enter (display,screen)
  exit string
  #);

glXQueryServerString:external
  (# display,screen,name:@integer;
     string: [0]@char
  enter (display,screen,name)
  exit string
  #);

glXQueryVersion:external
  (# display,major_ptr,minor_ptr:@integer;
     status:@boolean
  enter (display,major_ptr,minor_ptr)
  exit status
  #);

glXSwapBuffers:external (* 185 *)
  (# display:@integer;
     drawable:@integer;
  enter (display,drawable)
  #);

glXUseXFont:external
  (# font,first,count,listBase:@integer;
  enter (font,first,count,listBase)
  #);


glXWaitGL:external (* 205 *)
  (# #);

glXWaitX:external (* 207 *)
  (# #)
```

23.6 Agl Interface

```
ORIGIN 'opengl';

--lib:attributes--

(*------ GLX ---------- *)
(* note this is not complete,
 * only intendet for internal use,
 * as programs is not platform independent
 *)

AGL_NONE: (# exit 0 #);
AGL_ALL_RENDERERS: (# exit 1 #);  (* choose from all available renderers *)
AGL_BUFFER_SIZE: (# exit 2 #);  (* depth of the index buffer *)
AGL_LEVEL: (# exit 3 #);  (* level in plane stacking *)
AGL_RGBA: (# exit 4 #);  (* choose an RGBA format *)
AGL_DOUBLEBUFFER: (# exit 5 #);  (* double buffering supported *)
AGL_STEREO: (# exit 6 #);  (* stereo buffering supported *)
AGL_AUX_BUFFERS: (# exit 7 #);  (* number of aux buffers *)
AGL_RED_SIZE: (# exit 8 #);  (* number of red component bits *)
AGL_GREEN_SIZE: (# exit 9 #);  (* number of green component bits *)
AGL_BLUE_SIZE: (# exit 10 #);  (* number of blue component bits *)
AGL_ALPHA_SIZE: (# exit 11 #);  (* number of alpha component bits *)
AGL_DEPTH_SIZE: (# exit 12 #);  (* number of depth bits *)
AGL_STENCIL_SIZE: (# exit 13 #);  (* number of stencil bits *)
AGL_ACCUM_RED_SIZE: (# exit 14 #);  (* number of red accum bits *)
AGL_ACCUM_GREEN_SIZE: (# exit 15 #);  (* number of green accum bits *)
AGL_ACCUM_BLUE_SIZE: (# exit 16 #);  (* number of blue accum bits *)
AGL_ACCUM_ALPHA_SIZE: (# exit 17 #);  (* number of alpha accum bits *)

(*
 ** Extended attributes
 *)
AGL_OFFSCREEN: (# exit 50 #);  (* choose an off-screen capable renderer *)
AGL_FULLSCREEN: (# exit 51 #);  (* allow a full-screen renderer *)
AGL_MINIMUM_POLICY: (# exit 52 #);  (* never choose smaller buffers than requested *)
AGL_MAXIMUM_POLICY: (# exit 53 #);  (* choose largest buffers of type requested *)
AGL_CLOSEST_POLICY: (# exit 54 #);  (* choose the closest color buffer to request *)
AGL_PIXEL_SIZE: (# exit 55 #);  (* frame buffer bits per pixel *)
(* Renderer management *)
AGL_RENDERER_ID: (# exit 70 #);  (* request renderer by ID *)
AGL_SINGLE_RENDERER: (# exit 71 #);  (* choose a single renderer for all screens *)
(* OpenGL option capabilities *)
AGL_COMPLIANCE: (# exit 72 #);  (* bitfield of required opengl options *)
AGL_PERFORMANCE: (# exit 73 #);  (* bitfield of desired performance options *)
(* Only for aglDescribePixelFormat *)
AGL_VIRTUAL_SCREEN: (# exit 80 #);  (* virtual screen number *)
AGL_MULTISCREEN: (# exit 81 #);  (* single pixel format for multiple screens *)
AGL_BACKING_STORE: (# exit 82 #);  (* back buffer contents are valid after swap *)
AGL_PERFORMANCE_MSB: (# exit 83 #);  (* msb bitfield of performance indicators *)
AGL_PERFORMANCE_LSB: (# exit 84 #);  (* lsb bitfield of performance indicators *)


AGL_FULLSCREEN_640_480: (# exit 1 #);
AGL_FULLSCREEN_600_800: (# exit 2 #);


(*
 ** Property names for aglDescribeRenderer
 *)
                           (* AGL_OFFSCREEN: (# exit 50 #); *)
                           (* AGL_FULLSCREEN: (# exit 51 #); *)
                           (* AGL_RENDERER_ID: (# exit 70 #); *)
```

```
                            (* AGL_COMPLIANCE: (# exit 72 #); *)
                            (* AGL_MULTISCREEN: (# exit 81 #); *)
                            (* AGL_BACKING_STORE: (# exit 82 #); *)
                            (* AGL_PERFORMANCE_MSB: (# exit 83 #); *)
                            (* AGL_PERFORMANCE_LSB: (# exit 84 #); *)
AGL_BUFFER_MODES: (# exit 100 #);
AGL_MIN_LEVEL: (# exit 101 #);
AGL_MAX_LEVEL: (# exit 102 #);
AGL_TEXTURE_METHOD: (# exit 103 #);
AGL_POINT_AA_METHOD: (# exit 104 #);
AGL_LINE_AA_METHOD: (# exit 105 #);
AGL_POLYGON_AA_METHOD: (# exit 106 #);
AGL_FOG_METHOD: (# exit 107 #);
AGL_INDEX_SIZES: (# exit 108 #);
AGL_COLOR_MODES: (# exit 109 #);
AGL_ACCUM_MODES: (# exit 110 #);
AGL_DEPTH_SIZES: (# exit 111 #);
AGL_STENCIL_SIZES: (# exit 112 #);
AGL_MAX_AUX_BUFFERS: (# exit 113 #);

(*
 ** Renderer option bitfields
 *)
AGL_TRANSFORM: (# exit 0x00000001 #);    (* Geometry options *)
AGL_CLIPPING: (# exit 0x00000002 #);
AGL_LIGHTING: (# exit 0x00000004 #);

AGL_COPY_PIXELS: (# exit 0x00000008 #);    (* Primitive types *)
AGL_DRAW_PIXELS: (# exit 0x00000010 #);
AGL_READ_PIXELS: (# exit 0x00000020 #);
AGL_BITMAP: (# exit 0x00000040 #);
AGL_POINT: (# exit 0x00000080 #);
AGL_LINE: (# exit 0x00000100 #);
AGL_POLYGON: (# exit 0x00000200 #);
AGL_AA_POINT: (# exit 0x00000400 #);
AGL_AA_LINE: (# exit 0x00000800 #);
AGL_AA_POLYGON: (# exit 0x00001000 #);

AGL_SMOOTH_SHADING: (# exit 0x00002000 #);    (* Raster options *)
AGL_STIPPLE: (# exit 0x00004000 #);
AGL_TEXTURE: (# exit 0x00008000 #);
AGL_ALPHA_TEST: (# exit 0x00010000 #);
AGL_STENCIL_TEST: (# exit 0x00020000 #);
AGL_DEPTH_TEST: (# exit 0x00040000 #);
AGL_POLYGON_OFFSET: (# exit 0x00080000 #);
AGL_FOG: (# exit 0x00100000 #);
AGL_BLEND: (# exit 0x00200000 #);
AGL_DITHER: (# exit 0x00400000 #);
AGL_LOGIC_OP: (# exit 0x00800000 #);
AGL_MASKING: (# exit 0x01000000 #);

AGL_ALL_OPTIONS: (# exit 0x01ffffff #);

(*
 ** Renderer property bitfields
 *)

(* buffer_modes *)
AGL_STEREO_MODE: (# exit 0x00000001 #);
AGL_SINGLEBUFFER_MODE: (# exit 0x00000002 #);
AGL_DOUBLEBUFFER_MODE: (# exit 0x00000004 #);

(* algorithm methods *)
AGL_FASTEST: (# exit 0x00000002 #);
AGL_NICEST: (# exit 0x00000004 #);
```

```
(* bit depths *)
AGL_0_BIT: (# exit 0x00000001 #);
AGL_1_BIT: (# exit 0x00000002 #);
AGL_2_BIT: (# exit 0x00000004 #);
AGL_4_BIT: (# exit 0x00000008 #);
AGL_8_BIT: (# exit 0x00000010 #);
AGL_12_BIT: (# exit 0x00000020 #);
AGL_16_BIT: (# exit 0x00000040 #);
AGL_24_BIT: (# exit 0x00000080 #);
AGL_32_BIT: (# exit 0x00000100 #);
AGL_48_BIT: (# exit 0x00000200 #);
AGL_64_BIT: (# exit 0x00000400 #);

(* color modes *)
AGL_RGB_332: (# exit 0x00000001 #);  (* 8 rgb bit/pixel,     R=7:6, G=5:3, B=2:0 *)
AGL_ARGB_8332: (# exit 0x00000002 #);  (* 8−8 argb bit/pixel,  A=7:0, R=7:6, G=5:3, B=2:0 *)
AGL_RGB_555: (# exit 0x00000004 #);  (* 16 rgb bit/pixel,    R=14:10, G=9:5, B=4:0 *)
AGL_ARGB_1555: (# exit 0x00000008 #);  (* 16 argb bit/pixel,   A=15, R=14:10, G=9:5, B=4:0 *)
AGL_ARGB_8555: (# exit 0x00000010 #);  (* 8−16 argb bit/pixel, A=7:0, R=14:10, G=9:5, B=4:0 *)
AGL_RGB_888: (# exit 0x00000020 #);  (* 32 rgb bit/pixel,    R=23:16, G=15:8, B=7:0 *)
AGL_ARGB_8888: (# exit 0x00000040 #);  (* 32 argb bit/pixel,   A=31:24, R=23:16, G=15:8, B=7:0
AGL_RGB_161616: (# exit 0x00000080 #);  (* 32 rgb bit/pixel,    R=23:16, G=15:8, B=7:0 *)
AGL_ARGB_16161616: (# exit 0x00000100 #);  (* 32 argb bit/pixel,   A=31:24, R=23:16, G=15:8, B=
AGL_CL_4: (# exit 0x00000200 #);  (* 4 bit color look up table *)
AGL_CL_8: (# exit 0x00000400 #);  (* 8 bit color look up table *)
AGL_CL_16: (# exit 0x00000800 #);  (* 16 bit color look up table *)
(*
 ** Option names for aglSetOptions.
 *)
AGL_SWAP_HINT: (# exit 1 #);     (* Enable swap buffer hint rectangle *)
AGL_STATE_VALIDATION: (# exit 2 #);     (* Validate state for multi−screen functionality *)

(*
 ** Error return values from aglGetError.
 *)
AGL_NO_ERROR: (# exit 0 #); (* No error *)
AGL_BAD_ATTRIBUTE: (# exit 10000 #); (* unknown pixel format attribute *)
AGL_BAD_PROPERTY: (# exit 10001 #); (* unknown renderer property *)
AGL_BAD_PIXELFMT: (# exit 10002 #); (* invalid pixel fmt specified *)
AGL_BAD_RENDINFO: (# exit 10003 #); (* invalid renderer info specified *)
AGL_BAD_CONTEXT: (# exit 10004 #); (* invalid context specified *)
AGL_BAD_DRAWABLE: (# exit 10005 #); (* invalid drawable specified *)
AGL_BAD_ALLOC: (# exit 10006 #); (* memory allocation failure *)
AGL_BAD_GDEV: (# exit 10007 #); (* invalid GDevice for attribute *)
AGL_BAD_STATE: (# exit 10008 #); (* invalid context state *)
AGL_BAD_VALUE: (# exit 10009 #); (* invalid numerical value *)
AGL_BAD_MATCH: (# exit 10010 #); (* invalid share context match *)
AGL_GL_ERROR: (# exit 10011 #); (* GL error occurred *)

(*************************************************************************)



(*---------Various functions ---------------*)

(* Pixel format functions *)


aglChoosePixelFormat: external
  (# AGLGDHandle: @integer;
     ndev: @integer;
     attribs: @integer;
     AGLPixelFormat: @integer;
  enter  (AGLGDHandle, ndev, attribs)
  exit AGLPixelFormat
  #);
```

```
aglNextPixelFormat: external
  (# pix: @integer; (* AGLPixelFormat *)
     AGLPixelFormat: @integer;
  enter pix
  exit AGLPixelFormat
  #);

aglDescribePixelFormat: external
  (# AGLPixelFormat: @integer;
     attrib: @integer;
     value: @integer;
     bool: @Boolean;
  enter (AGLPixelFormat, attrib, value)
  exit bool
  #);


aglDevicesOfPixelFormat: external
  (# AGLPixelFormat: @integer;
     ndevs: @integer;
     AGLGDHandle: @integer;
  enter (AGLPixelFormat, ndevs)
  exit AGLGDHandle
  #);

(*  Context functions *)


aglCreateContext: external
  (# AGLPixelFormat: @integer;
     share: @integer; (* AGLContext *)
     AGLContext: @integer;
  enter (AGLPixelFormat, share)
  exit AGLContext
  #);

aglDestroyContext: external
  (# AGLContext: @integer;
     bool: @boolean;
  enter AGLContext
  exit bool
  #);


(* Current state functions *)

aglMakeCurrent: external
  (# AGLDrawable: @integer; (* This is just a pointer to a graphics port *)
     AGLContext: @integer;
     bool: @boolean;
  enter (AGLDrawable, AGLContext)
  exit bool
  #);

aglUpdateCurrent: external
  (# bool: @boolean;
  exit bool
  #);



(* Swapping *)

aglSwapBuffers: external
  (# bool: @boolean;
```

```
exit bool
#)
```

23.7 Glut Interface

```
ORIGIN 'opengl';

LIBFILE nti 'private/external/glut32/glut32.lib' default '' ;

-- lib: attributes --

GLUT_RGBA: (# exit 0 #);
GLUT_DOUBLE: (# exit 2 #);

GLUT_LEFT_BUTTON: (# exit 0 #);
GLUT_MIDDLE_BUTTON: (# exit 1 #);
GLUT_RIGHT_BUTTON: (# exit 2 #);

GLUT_DOWN: (# exit 0 #);
GLUT_UP: (# exit 1 #);

glutInit: external
  (# argcp: @integer;
     argv: @integer;
  enter (argcp, argv)
  do CallStd;
  #);

glutInitDisplayMode: external
  (# mode: @integer;
  enter mode
  do CallStd;
  #);
glutCreateWindow: external
  (# title: [1]@char;
     ID: @integer;
  enter title
  do CallStd;
  exit ID
  #);
glutMainLoop: external
  (#
  do CallStd;
  #);


glutReshapeWindow: external
  (# width, height: @integer;
  enter (width, height)
  do CallStd;
  #);

displayFunc: external
  (#
  do cExternalEntry;
     INNER;
  #);
idleFunc: external
  (#
  do cExternalEntry;
     INNER;
  #);
mouseFunc: external
  (# button: @integer;
     state: @integer;
     x: @integer;
     y: @integer;
```

```
  enter (button, state, x, y)
  do cExternalEntry;
     INNER;
  #);

motionFunc: external
  (# x, y: @integer;
  enter (x, y)
  do cExternalEntry;
     INNER;
  #);

keyboardFunc: external
  (# key: @char;
     x, y: @integer;
  enter (key, x, y)
  do cExternalEntry;
     INNER;
  #);
glutKeyboardFunc: external
  (# keyboard: ##keyboardFunc;
  enter keyboard##
  do CallStd;
  #);
glutMouseFunc: external
  (# mouse: ##mouseFunc;
  enter mouse##
  do CallStd;
  #);
glutMotionFunc: external
  (# motion: ##motionFunc;
  enter motion##
  do CallStd;
  #);
glutPassiveMotionFunc: external
  (# motion: ##motionFunc;
  enter motion##
  do CallStd;
  #);
glutDisplayFunc: external
  (# display: ##displayFunc;
  enter display##
  do CallStd;
  #);
reshapeFunc: external
  (# w, h: @integer;
  enter (w, h)
  do cExternalEntry;
     INNER;
  #);
glutReshapeFunc: external
  (# reshape: ##reshapeFunc;
  enter reshape##
  do CallStd;
  #);
glutIdleFunc: external
  (# idle: ##idleFunc;
  enter idle##
  do CallStd;
  #);
glutSwapBuffers: external
  (#
  do CallStd;
  #);
```

```
glutSetCursor: external
  (# ID: @integer;
  enter ID
  do CallStd;
  #);

GLUT_CURSOR_RIGHT_ARROW:                (# exit 0 #);
GLUT_CURSOR_LEFT_ARROW:                 (# exit 1 #);
GLUT_CURSOR_INFO:                       (# exit 2 #);
GLUT_CURSOR_DESTROY:                    (# exit 3 #);
GLUT_CURSOR_HELP:                       (# exit 4 #);
GLUT_CURSOR_CYCLE:                      (# exit 5 #);
GLUT_CURSOR_SPRAY:                      (# exit 6 #);
GLUT_CURSOR_WAIT:                       (# exit 7 #);
GLUT_CURSOR_TEXT:                       (# exit 8 #);
GLUT_CURSOR_CROSSHAIR:                  (# exit 9 #);
GLUT_CURSOR_UP_DOWN:                    (# exit 10 #);
GLUT_CURSOR_LEFT_RIGHT:                 (# exit 11 #);
GLUT_CURSOR_TOP_SIDE:                   (# exit 12 #);
GLUT_CURSOR_BOTTOM_SIDE:                (# exit 13 #);
GLUT_CURSOR_LEFT_SIDE:                  (# exit 14 #);
GLUT_CURSOR_RIGHT_SIDE:                 (# exit 15 #);
GLUT_CURSOR_TOP_LEFT_CORNER:            (# exit 16 #);
GLUT_CURSOR_TOP_RIGHT_CORNER:           (# exit 17 #);
GLUT_CURSOR_BOTTOM_RIGHT_CORNER:        (# exit 18 #);
GLUT_CURSOR_BOTTOM_LEFT_CORNER:         (# exit 19 #);
GLUT_CURSOR_INHERIT:                    (# exit 100 #);
GLUT_CURSOR_NONE:                       (# exit 101 #);
GLUT_CURSOR_FULL_CROSSHAIR:             (# exit 102 #)
```

Index

The entries in the alphabetic index consists of selected words and symbols from the body files of this manual − these are in **bold** font − as well as the identifiers defined in the public interfaces of the libraries − set in regular font.
In the manual, the entries, which can be found in the index are typeset like this. This can help localizing the identifier, when the link from the index if followed − especially in the case where the browser does not scroll the line to the top, e.g. because there is less than a page of text left.
In the small table of letters and symbols below, each entry links directly to the section of the index containing entries starting with the corresponding letter or symbol.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

B

C

D

E

F

G

GL_CURRENT_RASTER_POSITION_VALID
GL_CURRENT_RASTER_TEXTURE_COORDS
GL_CURRENT_TEXTURE_COORDS
GL_CW
GL_DECAL
GL_DECR
GL_DEPTH
GL_DEPTH_BIAS
GL_DEPTH_BITS
GL_DEPTH_BUFFER_BIT
GL_DEPTH_CLEAR_VALUE
GL_DEPTH_COMPONENT
GL_DEPTH_FUNC
GL_DEPTH_RANGE
GL_DEPTH_SCALE
GL_DEPTH_TEST
GL_DEPTH_WRITEMASK
GL_DIFFUSE
GL_DISTANCE_ATTENUATION_EXT
GL_DITHER
GL_DOMAIN
GL_DONT_CARE
GL_DOUBLE
GL_DOUBLEBUFFER
GL_DRAW_BUFFER
GL_DRAW_PIXEL_TOKEN
GL_DST_ALPHA
GL_DST_COLOR
GL_EDGE_FLAG
GL_EDGE_FLAG_ARRAY
GL_EDGE_FLAG_ARRAY_COUNT_EXT
GL_EDGE_FLAG_ARRAY_EXT
GL_EDGE_FLAG_ARRAY_POINTER
GL_EDGE_FLAG_ARRAY_POINTER_EXT
GL_EDGE_FLAG_ARRAY_STRIDE
GL_EDGE_FLAG_ARRAY_STRIDE_EXT
GL_EMISSION
GL_ENABLE_BIT
GL_EQUAL
GL_EQUIV
GL_EVAL_BIT
GL_EXP
GL_EXP2
GL_EXTENSIONS
GL_EYE_LINEAR
GL_EYE_PLANE
GL_FALSE
GL_FASTEST
GL_FEEDBACK
GL_FEEDBACK_BUFFER_POINTER
GL_FEEDBACK_BUFFER_SIZE
GL_FEEDBACK_BUFFER_TYPE
GL_FILL
GL_FLAT
GL_FLOAT

glEvalCoord1d
glEvalCoord1dv
glEvalCoord1f
glEvalCoord1fv
glEvalCoord2d
u
v
glEvalCoord2dv
u
glEvalCoord2f
glEvalCoord2fv
glEvalMesh1
i1
i2
mode
glEvalMesh2
i1
i2
j1
j2
mode
glEvalPoint1
i
glEvalPoint2
i
j
glFeedbackBuffer
buffer
size
type
glFinish
glFlush
glFogf
param
pname
glFogfv
params
pname
glFogi
param
pname
glFogiv
params
pname
glFrontFace
mode
glFrustum
bottom
far_val
left
near_val
right
top
glGenLists
range
result
glGenTextures

GLU_INTERIOR
GLU_INVALID_ENUM
GLU_INVALID_VALUE
GLU_LINE
GLU_MAP1_TRIM_2
GLU_MAP1_TRIM_3
GLU_NONE
GLU_NURBS_ERROR1
GLU_NURBS_ERROR10
GLU_NURBS_ERROR11
GLU_NURBS_ERROR12
GLU_NURBS_ERROR13
GLU_NURBS_ERROR14
GLU_NURBS_ERROR15
GLU_NURBS_ERROR16
GLU_NURBS_ERROR17
GLU_NURBS_ERROR18
GLU_NURBS_ERROR19
GLU_NURBS_ERROR2
GLU_NURBS_ERROR20
GLU_NURBS_ERROR21
GLU_NURBS_ERROR22
GLU_NURBS_ERROR23
GLU_NURBS_ERROR24
GLU_NURBS_ERROR25
GLU_NURBS_ERROR26
GLU_NURBS_ERROR27
GLU_NURBS_ERROR28
GLU_NURBS_ERROR29
GLU_NURBS_ERROR3
GLU_NURBS_ERROR30
GLU_NURBS_ERROR31
GLU_NURBS_ERROR32
GLU_NURBS_ERROR33
GLU_NURBS_ERROR34
GLU_NURBS_ERROR35
GLU_NURBS_ERROR36
GLU_NURBS_ERROR37
GLU_NURBS_ERROR4
GLU_NURBS_ERROR5
GLU_NURBS_ERROR6
GLU_NURBS_ERROR7
GLU_NURBS_ERROR8
GLU_NURBS_ERROR9
GLU_OUT_OF_MEMORY
GLU_OUTLINE_PATCH
GLU_OUTLINE_POLYGON
GLU_OUTSIDE
GLU_PARAMETRIC_ERROR
GLU_PARAMETRIC_TOLERANC
GLU_PATH_LENGTH
GLU_POINT
GLU_SAMPLING_METHOD
GLU_SAMPLING_TOLERANCE
GLU_SILHOUETTE

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z