4-27: Lidskjalv: User Interface Framework - Reference Manual

Table of Contents

Copyright Notice **1**

Overview of Lidskjalv **3**

The Lidskjalv Libraries **5**

Utilities Libraries **7**

Multimedia Libraries **8**

OpenGL Libraries **9**

Chapter 1: The guienv Library **10**

Event handling **12**

Menu facilities **13**

Window facilities **15**

Using the guienv Library **17**

Examples of Use of the guienv Fragment **19**
simplewindow.bet 19
windowWithStandardMenubar.bet 20

Chapter 2: The stddialogs Library **22**

Using the stddialogs Library **23**

Examples of Use of the stddialogs Fragment **24**
file.bet 24

Chapter 3: The controls Library **26**

Using the controls Library **30**

Examples of Use of the controls Fragment **31**
button.bet 31

Chapter 4: The fields Library **34**

Using the fields Library **35**

Examples of Use of the fields Fragment **36**
texteditor.bet 36

Chapter 5: The figureitems Library **38**

Using the figureitems Library **39**

Table of Contents

Examples of Use of the figureitems Fragment.....40
draw.bet 40

Chapter 6: The scrolllists Library 43

Using the scrolllists Library 44

Examples of Use of the scrolllists Fragment.....45
textscrolllist.bet 45

Chapter 7: The graphmath Library 47

Using the graphmath Library 48

Examples of Use of the graphmath Fragment.....49
containspoint.bet 49

Chapter 8: The graphics Library 51

Using the graphics Library 52

Examples of Use of the graphics Fragment.....53
drawbitmap.bet 53

Chapter 9: The styledtext Library 55

Using the styledtext Library 56

Examples of Use of the styledtext Fragment.....57

Chapter 10: The guienvactions Library 58

Using the guienvactions Library 59

Examples of Use of the guienvactions Fragment.....60
Using keyboard actions 60
keyboardactions.bet 60

Chapter 11: The controlactions Library 62

Using the controlactions Library 63

Examples of Use of the controlactions Fragment.....64
buttonactions.bet 64

Chapter 12: The fieldsactions Library 66

Using the fieldsactions Library 67

Examples of Use of the fieldsactions Fragment.....68
textfieldactions.bet 68

Table of Contents

Chapter 13: The guienvall Library 70

Using the guienvall Library 71

Chapter 14: The guienvsystemenv Library 72

Using the guienvsystemenv Library 73

Appendix A: Demo Programs.....74

Appendix B: Implementation Design 76

References 83

Controls Interface 84

Controlsactions Interface 90

Fields Interface 92

Fieldsactions Interface 98

Figureitems Interface 99

Graphics Interface 103

Graphmath Interface 106

Guienv Interface 113

Guienvactions Interface 139

Guienvall Interface 141

Guienvsystemenv Interface 142

Table of Contents 143

Index of Identifiers 145

A.....145
B.....145
C.....146
D.....147
E.....148
F.....148
G.....149
H.....155
I.....155
K.....156
L.....156
M.....157

Table of Contents

N....158
O....158
P....159
Q....160
R....160
S....161
T....164
U....165
V....165
W....166
X....166
Y....166

Scrolllists Interface....167

Stddialogs Interface....171

Styledtext Interface....173

Copyright Notice

Mjłlner Informatics Report
MIA 94-27
August 1999

Copyright ' 1994-99 Mjłlner Informatics.
All rights reserved.
No part of this document may be copied or distributed
without the prior written permission of Mjłlner Informatics

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.

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.

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.

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.

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.

Chapter 1: 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.

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.

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.

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:˚onVisibilityChanged, 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).

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.

Chapter 1: The guienv Library

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).

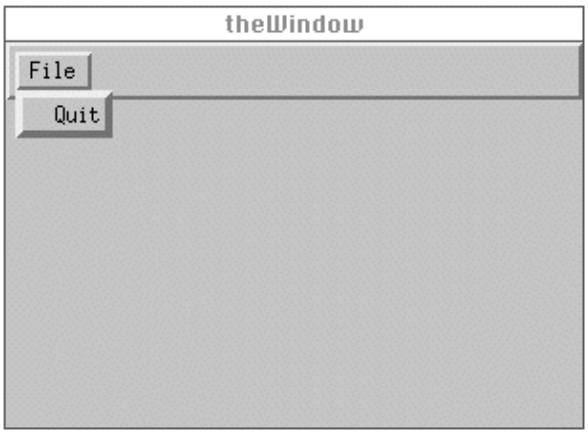
simplewindow.bet

```
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.

windowWithStandardMenubar.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
#)
```



Lidskjalv - Reference Manual

' Mjölner Informatics

Chapter 2: 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.

Using the stddialogs Library

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

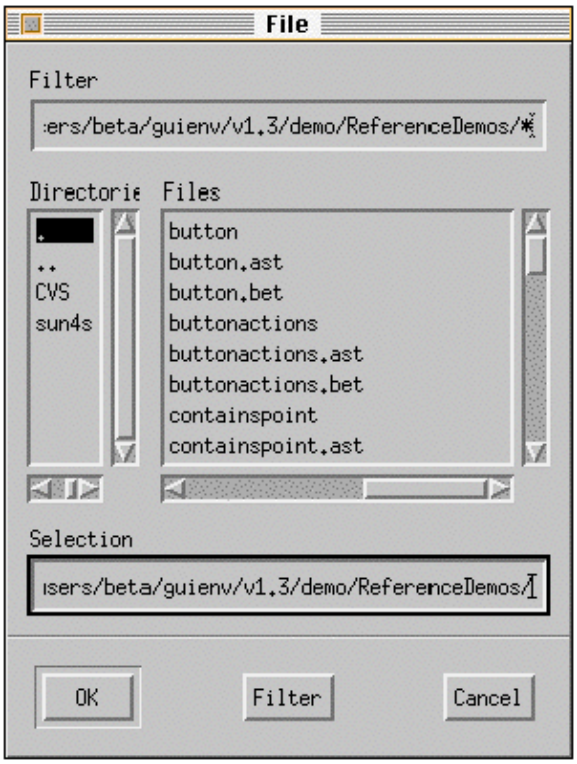
```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/stddialogs'
--- program: descriptor ---
guienv(# ...
      do ...
         ... -> fileSelectionDialog -> ...;
         ...
      #)
```

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.

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
#)
```



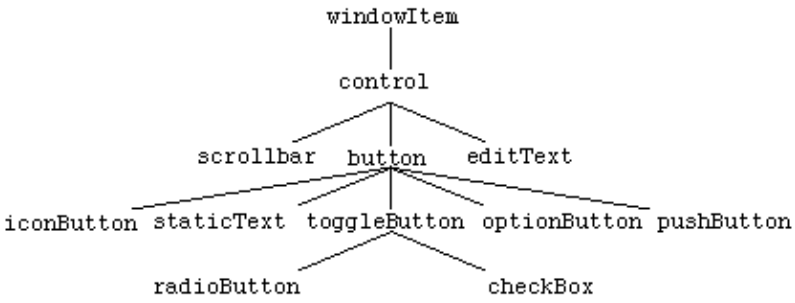
Lidskjalv - Reference Manual

' Mjølner Informatics

Chapter 3: 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).

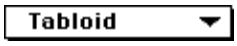
editText

Used to allow the user to enter some text.

pushButton

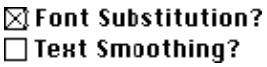
A button is used to specify some actions to be taken.

optionButton



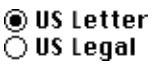
Used to specify a button with associated pop-up menu.

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.

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;
         ...
      #)
```

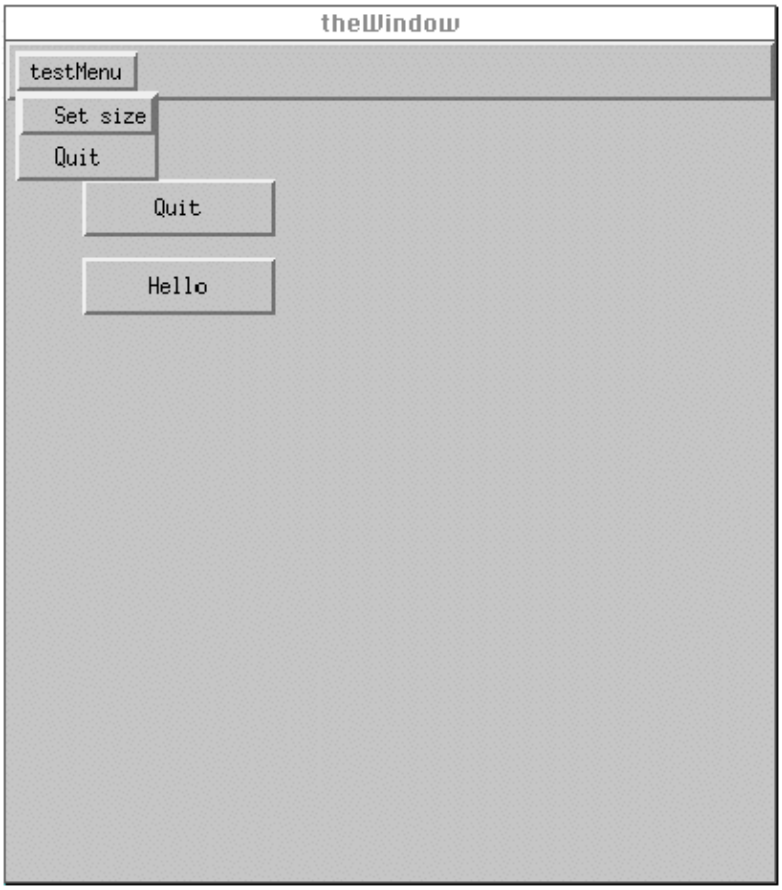
Examples of Use of the controls Fragment

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

button.bet

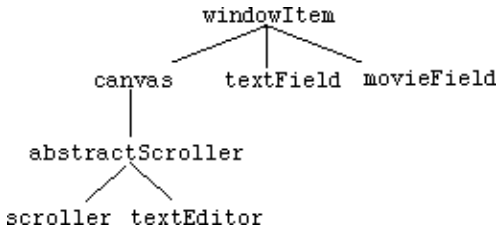
```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/controls';
(* This demo shows how to create a window with two pushButtons, and
 *  how to give a button a new size on runtime.
 *)
--- program: descriptor ---
guienv
(# theWindow: @window
     (# menubarType::
          (# testMenu: @menu
               (# 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
#)
```



Chapter 4: 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).

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;
         ...
      #)
```

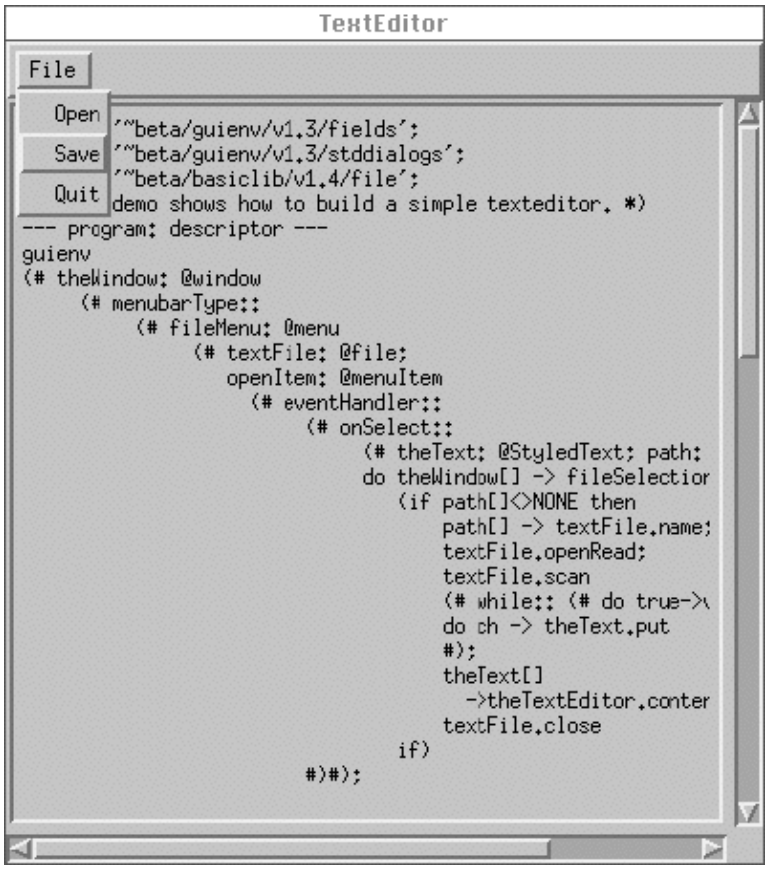
Examples of Use of the fields Fragment

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

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;
```

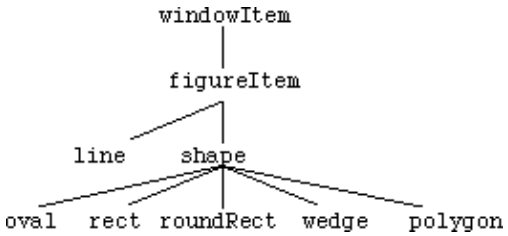
```
                        quitItem.open; quitItem[] -> append
                  #)#);
               open::
                  (# do fileMenu.open; fileMenu[] -> append #)
            #); (* menubarType *)
         eventhandler::
            (# onAboutToClose:: (# do terminate #) #);
         thetextEditor: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



Chapter 5: 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.

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;
         ...
      #)
```

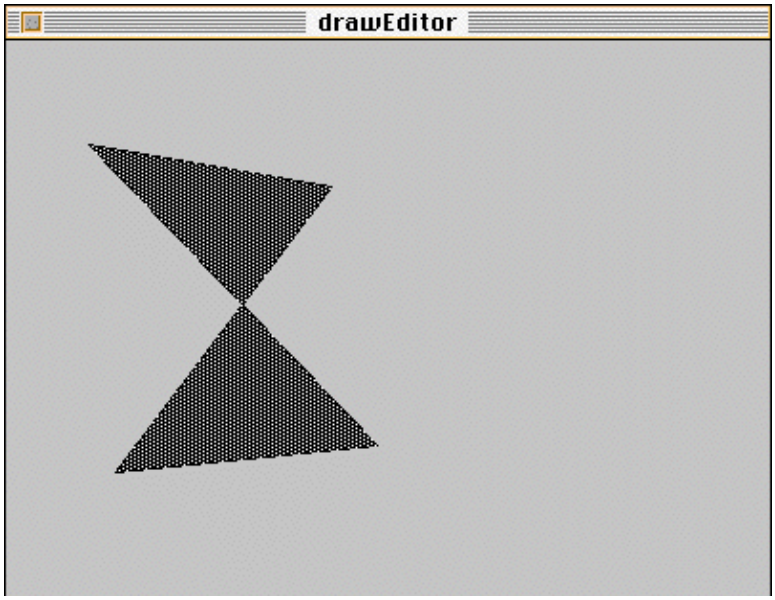
Examples of Use of the figureitems Fragment

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

draw.bet

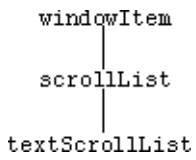
```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/figureitems';
(* This demo gives an example of how you can draw lines and polygons
 * with mouse using the figureitems "line" and "polygon".  Clicking
 * with the left mouse button defines a node in the polygon/line,
 * clicking with the right mouse 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

Chapter 6: 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.

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 -> ...;
         ...
      #)
```

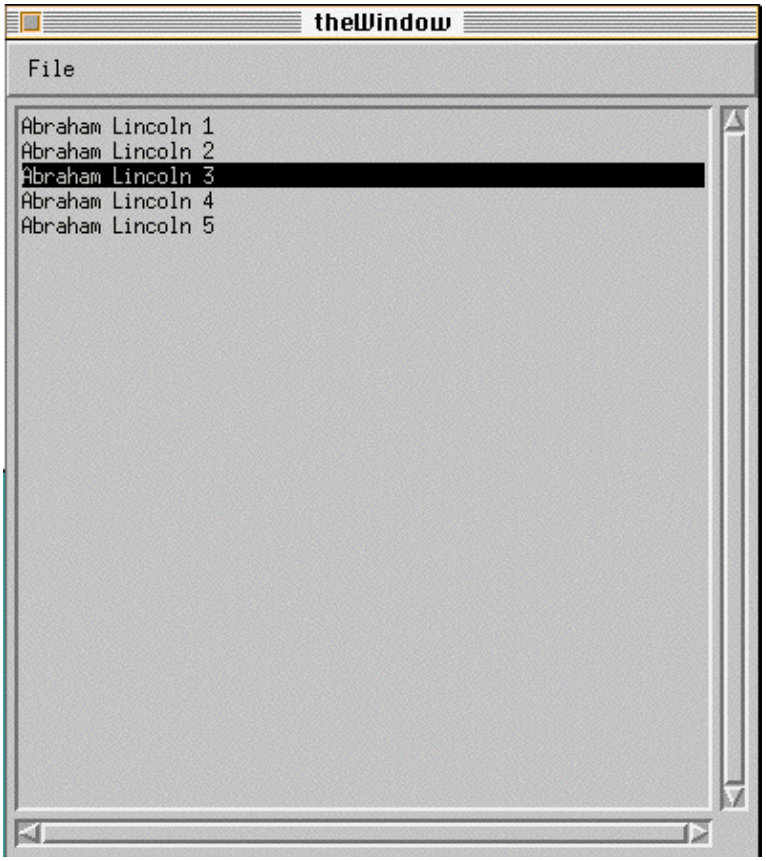
Examples of Use of the scrolllists Fragment

This example illustrates how to use the textscrolllist windowItem.

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
#)
```



Chapter 7: 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.

Using the graphmath Library

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

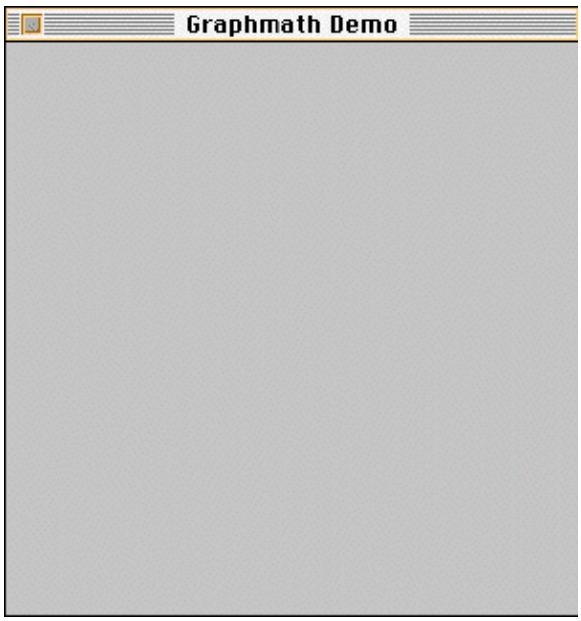
```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/graphmath'
--- program: descriptor ---
guienv(# p1, p2: @point;
         ...
      do ...
         p1 -> p2.inset -> ...;
         ...
      #)
```

Examples of Use of the graphmath Fragment

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

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
#)
```



Chapter 8: 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.

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;
         ...
      #)
```

Examples of Use of the graphics Fragment

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

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
#)
```

Chapter 9: 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.

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;
         ...
      #)
```

Examples of Use of the styledtext Fragment

No demoes, since this extension is not implemented yet.

Chapter 10: 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).

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;
         ...
      #)
```

Examples of Use of the guienvactions Fragment

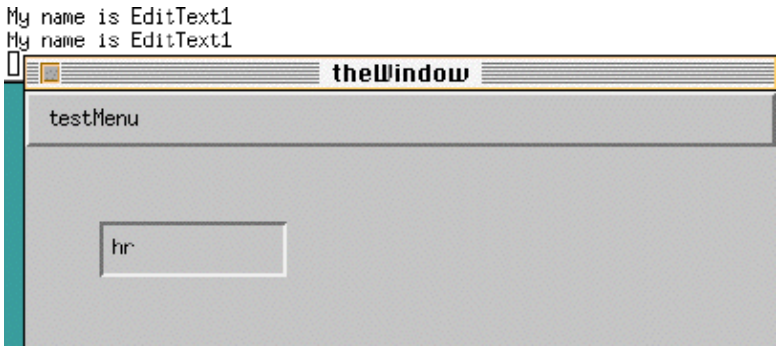
Using keyboard actions

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

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
#)
```



Chapter 11: The controlactions Library

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

Using the controlactions Library

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

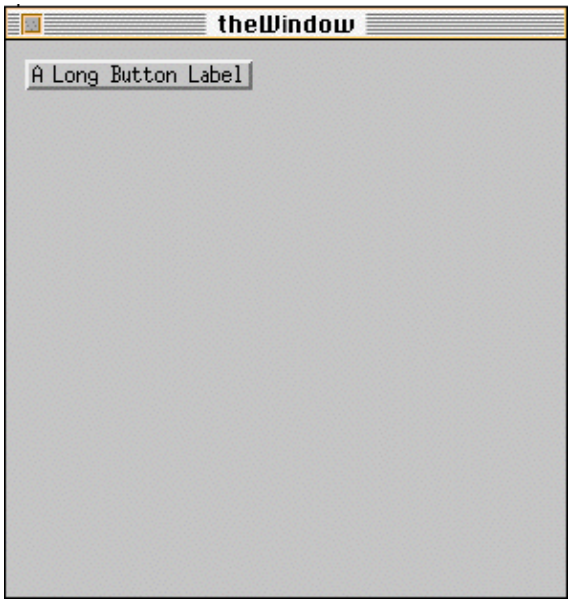
```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/controlactions'
--- program: descriptor ---
guienv(# lca: @labelChangedAction;
         ...
      do ...
         lca[] -> appendAction;
         ...
      #)
```

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.

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
#)
```


theWindow
A Long Button Label

Chapter 12: The fieldsactions Library

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

Using the fieldsactions Library

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

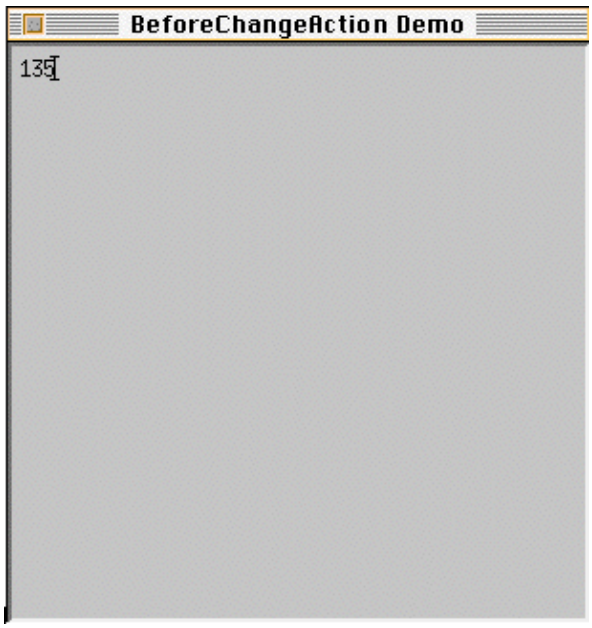
```
ORIGIN '~beta/guienv/guienv';
INCLUDE '~beta/guienv/fieldsactions'
--- program: descriptor ---
guienv(# bca: @beforeChangeAction;
         ...
      do ...
         bca[] -> appendAction;
         ...
      #)
```

Examples of Use of the fieldsactions Fragment

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

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
#)
```


BeforeChangeAction Demo
135

Chapter 13: 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.

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;
         ...
      #)
```

Chapter 14: 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.

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 (# ... #);
   ...
#)
```

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.
- windowWithStandardMenubar.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.

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

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

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>>;
    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>>;
     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>>;
    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[] #)
  #)
```

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
  #)
```

Controlsactions Interface ' Mjłlner Informatics

Fields Interface

```
ORIGIN 'guienv';
LIB_DEF 'guienvfields' '../lib';
INCLUDE 'controls';
INCLUDE 'styledtext';
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>>;
     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 true -> allow;
           INNER;
        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: @...;
  #)
```

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;
  #)
```

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: @...;
#)
```

Figureitems Interface ' Mjølner Informatics

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 ...;
#)
```

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[] #)
  #)
```

Guienv Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'guienv' '../lib';
INCLUDE '~beta/containers/list';
INCLUDE 'graphmath';
INCLUDE 'keys';
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
       #);
     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.
            * Only one modifier at a time is supported!
            *)
           (# 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
  (# 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>>;
   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>>;
     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.
   *)
(# 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
        #);
      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
   ...
   #);
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 *);

(* 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: @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;
...
#)
```

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
  #)
```

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';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1991-96
 *       All rights reserved.
 *)
```

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 *)
---]]
```

Table of Contents

controls Interface

controlsactions Interface

fields Interface

fieldsactions Interface

figureitems Interface

graphics Interface

graphmath Interface

guienv Interface

guienvactions Interface

guienvall Interface

guienvsystemenv Interface

scrolllists Interface

stddialogs Interface

styledtext Interface

Index

Interface Descriptions:
Contents

' Mjölner Informatics

JavaScript Required!

It appears, that either your browser does not support JavaScript, or you have disabled scripting.
The links to identifiers below will not work correctly without JavaScript.
They will, however, take you to the correct file. But you must use your browser's Find facility to get to the correct location within the page.

Index of Identifiers

A

a [2] [3]
aboutToClose
aboutToCloseAction
abstractScroller
  close
  contents
  contentsType
  open
  private
  scroll
action [2]
activate
activateAction
add [2]
alertUser
all
allocate
altKey
andBlend
angle [2]
append [2] [3]
appendAction
appendMenubar
applicationMenubar
arrow
ascent
attach
availableSizes

B

b
backgroundColor [2] [3] [4] [5]
basicEvent
basicEventAction
beep
beforeChange
beforeChangeAction
bifrostprivate
bindBottom
bindLeft
bindRight
bindTop
black
blue
bold
border
borderStyleChanged
borderStyleChangedAction
borderStyles
borderVisibleChanged
borderVisibleChangedAction
bottom
bottomRight
bounds

bringBack
bringBehind
bringToFront
busyCursor
button
  close
  eventhandler
    labelChanged
    onFrameChanged
    onLabelChanged
    onMouseDown
    onRefresh
    onStyleChanged
    styleChanged
  foregroundColor
  label
  open
  private
  style
buttonDown
buttonDownAction
buttonState [2]
buttonUp
buttonUpAction

C

c
canvas
cb
ch
checkBox
  close
  open
  private
checked
childFrameChanged
childFrameChangedAction
circleAngle
  a
  angle
  cos_a
  cx
  cy
  sin_a
  x
  y
clear [2] [3] [4] [5] [6]
clearContents
clearMenuitem
clipboard
close [2] [3] [4] [5] [6] [7] [8] [9]
  [10] [11] [12] [13] [14] [15] [16] [17] [18] [19]
  [20] [21] [22] [23] [24] [25]
closeMenuitem
color
containsPoint [2]
containsRectangle
contents [2] [3] [4] [5] [6]
contentsType [2] [3]
control
  close
  eventHandler

  open
  private
controlKey
copy [2]
copyMenuitem
cos_a [2]
cross
curPt
currentItem
currentItemChanged
currentItemChangedAction
cursor
cursors
cursorType
cut
cutMenuitem
cx [2]
cy [2]

D

d
deactivate
deactivateAction
defaultButton
  theButton
defaultStyle
delay
delete [2] [3] [4]
deleteAction
deleteFirst
deleteLast
deleteMenubar
descent
deselect
detach
dialog [2]
difference
disable [2]
disableEventType
disableTarget
disableTargetAction
displaywarnings
dispose [2]
dkGray
doSetup
doTrace
doubleClick
doubleDashedLine
doubleLine
drag
draw
drawLine
drawOval
drawPixmap
drawPolygon
drawRaster
drawRect
drawRoundRect
drawSlice
drawSpot
drawSpots
drawText

drawTo
dynamicMenuitem

E

editMenu
editText
  close
  contents
  eventhandler
    onDisableTarget
    onEnableTarget
    onFrameChanged
    onKeyDown
    onMouseDown
    onRefresh
  open
  private
  style
empty [2]
enable [2]
enabled [2]
enabledChanged
enabledChangedAction
enableEventType
enableTarget
enableTargetAction
end [2]
endAngle
erase
etchedIn [2]
etchedOut [2]
event
eventhandler [2] [3] [4] [5] [6] [7] [8] [9]
  [10] [11]
eventHandler
eventhandler
eventHandler
eventhandler [2] [3] [4] [5] [6] [7] [8] [9]
  [10] [11] [12] [13] [14]

F

face
father
fatherFrameChanged
fatherFrameChangedAction
figureItem
  eventhandler
    onRefresh
  open
  pen
    backgroundColor
    foregroundColor
    size
    stipple
  private
fileCreationDialog
fileDialog
fileMenu
fileName

fileSelectionDialog
fill
fillOval
fillPolygon
fillRect
fillRoundRect
fillSlice
first
fitToContents
foregroundColor [2] [3] [4]
frame [2]
frameChanged
frameChangedAction

G

get
getChar
getInverse
getItemRectangle
getMenuitemByNumber
getText
global [2]
globalPosition [2]
globalToLocal
graphics
  draw
  drawLine
  drawOval
  drawPolygon
  drawRaster
  drawRect
  drawRoundRect
  drawSlice
  drawSpot
  drawSpots
  drawText
  drawTo
  fillOval
  fillPolygon
  fillRect
  fillRoundRect
  fillSlice
  move
  moveTo
  overrideChildren
  pen
    backgroundColor
    foregroundColor
    mode
    size
    stipple
  private
  style
gray
green
GUIenv
  applicationMenubar
    theMenubar
  beep
  bifrostprivate
  borderStyles
    etchedIn

  etchedOut
  shadowIn
  shadowOut
  simple
clipboard
  clearContents
  hasText
  textContents
color
  blue
  green
  red
cursor
  private
cursors
  arrow
  cross
  iBeam
  plus
  watch
displaywarnings
doSetup
doTrace
guienvWait
  ticks
interfaceObject
  action
  appendAction
  close
  deleteAction
  disableEventType
  enableEventType
  eventhandler
    activate
    basicEvent
      altKey
      buttonState
      controlKey
      globalPosition
      localPosition
      metaKey
      shiftKey
      when
    deactivate
    event
    keyDown
    keyEvent
      ch
      key
    mouseDown
      delay
    mouseEvent
      doubleClick
    mouseUp
    onActivate
    onDeactivate
    onKeyDown
    onMouseDown
    onMouseUp
    onRefresh
    refresh
  interfaceObjectException
  notOpenedError
  notOpenedException
  open
  prependAction

  private
  theEventhandler
menu
  append
  clear
  close
  delete
  disable
  dynamicMenuitem
    attach
    detach
    eventhandler
      onSelect
      onStatus
    theAction
  enable
  enabled
  eventhandler
    onSelect
    select
  getMenuitemByNumber
  menuAction
    onSelect
    onStatus
    theMenuitem
  menuitem
    checked
    eventhandler
      onSelect
      onStatus
      select
    key
    name
    open
    position
    private
    specialkey
    subMenu
  name
    theName
  noOfMenuitems
  open
  popUp
  private
  scan
  separator
menubar
  append
  appendMenubar
  clear
  close
  delete
  deleteMenubar
  open
  private
  replaceMenubar
  scan
menubarType
mouse
  busyCursor
  buttonState
  globalPosition
onOpenDocument
  fileName
onQuit
  okToQuit

onStartApplication
patterns
  black
  dkGray
  gray
  ltGray
  white
pixmap
  clear
  dispose
  drawPixmap
  height
  init
  mask
  private
  read
  transparent
  transparentColor
  width
private
screenRectangle
  theRectangle
screenRgn
  rgn
separatorStyles
  doubleDashedLine
  doubleLine
  etchedIn
  etchedOut
  singleDashedLine
  singleLine
setupDone
specKeys
standardMenubar
  editMenu
  fileMenu
  open
  standardEditMenu
    clearMenuitem
    copyMenuitem
    cutMenuitem
    open
    pasteMenuitem
    undoMenuitem
  standardFileMenu
    closeMenuitem
    newMenuitem
    open
    openMenuitem
    pageSetUpMenuitem
    printMenuitem
    quitMenuitem
    revertMenuitem
    saveAsMenuitem
    saveMenuitem
  theEditMenu
  theFileMenu
standardTextStyle
terminate
textFaces
  bold
  italic
  plain
textStyle
  ascent
  availableSizes

descent
face
leading
lineHeight
maxChWidth
name
private
size
widthOfChar
widthOfText
timer
action
once
private
start
stop
trace
transferModes
andBlend
copy
erase
invertCopy
notAndBlend
notOrBlend
orBlend
xorBlend
warnStream
window
backgroundColor
bringBack
bringBehind
bringToFront
canvas
close
eventhandler
childFrameChanged
onActivate
onChildFrameChanged
onDeactivate
onFrameChanged
onMouseDown
onMouseUp
onRefresh
onVisibleChanged
open
private
scan
selection
add
clear
empty
remove
scan
set
close
contents
eventhandler
aboutToClose
onAboutToClose
onActivate
onDeactivate
frame
globalToLocal
global
local
hide

insideRectangle
localToGlobal
  global
  local
maxSize
menubarType
menubarVisible
minSize
open
position
private
resizeable
separator
  close
  eventhandler
    onRefresh
    onStyleChanged
    styleChanged
  open
  private
  style
  vertical
show
showModal
size
target
theMenubar
title
type
update
visible
windowitem
  backgroundColor
  bindBottom
  bindLeft
  bindRight
  bindTop
  border
    style
    visible
  close
  cursorType
  disable
  drag
  enable
  enabled
  eventhandler
    borderStyleChanged
    borderVisibleChanged
    disableTarget
    enabledChanged
    enableTarget
    fatherFrameChanged
    frameChanged
    hiliteChanged
    mouseEnter
    mouseLeave
    onBorderStyleChanged
    onBorderVisibleChanged
    onDisableTarget
    onEnabledChanged
    onEnableTarget
    onFatherFrameChanged
    onFrameChanged
    onHiliteChanged
    onMouseEnter

onMouseLeave
onRefresh
onTheCursorChanged
onVisibleChanged
theCursorChanged
visibleChanged
father
fitToContents
frame
hide
hilite
insideRectangle
move
open
position
private
resize
show
size
theCursor
trackMouse
curPt
mouseMove
mousePress
mouseRelease
prevPt
update
visible
windowTypes
dialog
modelessDialog
normal
palette
XsystemEnvHandleCallback
XsystemEnvHandleCallbackP
cb
XsystemEnvPresent
guienvWait

H

h
has
hasText
height
hide [2]
hilite
hiliteChanged
hiliteChangedAction
hr

I

iBeam
icon
iconButton
close
eventhandler
iconChanged
onHiliteChanged
onIconChanged

onMouseDown
onRefresh
onShowLabelChanged
showLabelChanged
icon
open
private
showLabel
iconChanged
iconChangedAction
ID
IDmatrix
ID
init
insert [2]
inset [2]
insideRectangle [2]
interfaceObject
interfaceObjectException
intersection [2]
inverse
inverseTransformPoint
inverseTransformRectangle
invertCopy
isEmpty [2]
isEqual [2] [3]
isOneStyle
italic
itemHeight
itx
ity

K

key [2]
keyDown
keyDownAction
keyEvent
keyEventAction

L

label
labelChanged
labelChangedAction
last
leading
left
length [2]
line
end
eventhandler
onFrameChanged
onHiliteChanged
onRefresh
open
private
start
lineHeight
local [2]
localPosition

localToGlobal
ltGray

M

margin
mask
matrix
  a
  b
  c
  d
  getInverse
  inverse
  inverseTransformPoint
  inverseTransformRectangle
  mult
  transformPoint
  transformRectangle
  tx
  ty
maxChWidth
maxSize
maxValue
maxValueChanged
maxValueChangedAction
menu
menuAction
menubar
menubarType [2]
menubarVisible
menuitem
messageDialog
metaKey
minSize
mode
modelessDialog
mouse
mouseDown
mouseDownAction
mouseEnter
mouseEvent
mouseEventAction
mouseLeave
mouseMove
mousePress
mouseRelease
mouseUp
mouseUpAction
move [2]
moveMatrix
  itx
  ity
moveTo
movie
movieField
  close
  contents
  open
  private
  scaleToFit
mult
multipleSelection

N

name [2] [3]
newMenuitem
noOfMenuitems
normal
notAndBlend
noteUser
notOpenedError
notOpenedException
notOrBlend
numberOfItems

O

offset [2]
okToQuit
onAboutToClose
onActivate [2] [3] [4] [5]
onBeforeChange
onBorderStyleChanged
onBorderVisibleChanged
onButtonDown
onButtonUp
once
onChildFrameChanged
onCurrentItemChanged
onDeactivate [2] [3] [4] [5]
onDisableTarget [2] [3]
onEnabledChanged
onEnableTarget [2] [3]
onFatherFrameChanged
onFrameChanged [2] [3] [4] [5] [6] [7] [8] [9]
[10]
onHiliteChanged [2] [3] [4]
onIconChanged
onKeyDown [2] [3]
onLabelChanged [2]
onMaxValueChanged
onMouseDown [2] [3] [4] [5] [6] [7] [8]
onMouseEnter
onMouseLeave
onMouseUp [2] [3] [4]
onOpenDocument
onPageDown
onPageScrollAmountChanged
onPageUp
onPopUpMenuChanged
onQuit
onRefresh [2] [3] [4] [5] [6] [7] [8] [9]
[10] [11] [12] [13] [14] [15] [16] [17] [18] [19]
[20]
onScrollAmountChanged
onSelect [2] [3] [4] [5]
onShowLabelChanged
onStartApplication
onStateChanged
onStatus [2] [3]
onStyleChanged [2] [3]
onTextChanged
onTheCursorChanged

onThumbMoved
onValueChanged
onVisibleChanged [2] [3]
open [2] [3] [4] [5] [6] [7] [8] [9]
[10] [11] [12] [13] [14] [15] [16] [17] [18] [19]
[20] [21] [22] [23] [24] [25] [26] [27] [28] [29]
[30] [31] [32] [33] [34] [35] [36] [37]
openMenuitem
optionButton
close
currentItem
eventhandler
currentItemChanged
onCurrentItemChanged
onLabelChanged
onPopUpMenuChanged
onRefresh
onStyleChanged
popUpMenuChanged
open
popupMenu
private
orBlend
oval
eventhandler
onRefresh
open
ovalAngle
a
angle
cos_a
cx
cy
hr
sin_a
vr
x
y
overrideChildren

P

pageDown
pageDownAction
pageScrollAmount
pageScrollAmountChanged
pageScrollAmountChangedAction
pageSetUpMenuitem
pageUp
pageUpAction
palette
paste
pasteMenuitem
patterns
pen [2]
pixmap
plain
plus
point
add
h
isEqual
subtract

v
points
polygon
eventhandler
onFrameChanged
onRefresh
open
points
private
popUp
popupMenu
popUpMenuChanged
popUpMenuChangedAction
position [2] [3]
posToPt
prepend
prependAction
prevPt
printMenuitem
private [2] [3] [4] [5] [6] [7] [8] [9]
[10] [11] [12] [13] [14] [15] [16] [17] [18] [19]
[20] [21] [22] [23] [24] [25] [26] [27] [28] [29]
[30] [31] [32] [33] [34] [35] [36] [37] [38]
pToAngle
ptToPos
pushButton
close
eventHandler
open
private

Q

quitMenuitem

R

radioButton
close
open
private
read
rect
eventhandler
onRefresh
open
rectangle
bottom
bottomRight
containsPoint
inset
intersection
isEmpty
isEqual
left
offset
pToAngle
right
set
setFromPoints
size

top
topLeft
union
red
refresh
refreshAction
region
allocate
bounds
containsPoint
containsRectangle
difference
dispose
empty
inset
intersection
isEmpty
isEqual
offset
private
setFromRectangle
symDiff
union
remove
replaceMenubar
resize
resizeable
revertMenuitem
rgn
right
rotateMatrix
theta
roundness
roundRect
eventhandler
onRefresh
open
private
roundness

S

saveAsMenuitem
saveMenuitem
scaleMatrix
scaleToFit
scan [2] [3] [4] [5]
scanSelection
scanText
scanTextWithStyle
screenRectangle
screenRgn
scroll [2] [3]
scrollAmount
scrollAmountChanged
scrollAmountChangedAction
scrollbar
close
eventhandler
buttonDown
buttonUp
maxValueChanged
onActivate

onButtonDown
onButtonUp
onDeactivate
onFrameChanged
onMaxValueChanged
onMouseDown
onPageDown
onPageScrollAmountChanged
onPageUp
onRefresh
onScrollAmountChanged
onThumbMoved
onValueChanged
pageDown
pageScrollAmountChanged
pageUp
scrollAmountChanged
thumbMoved
valueChanged
length
maxValue
open
pageScrollAmount
private
scrollAmount
value
vertical
scroller
close
contentsType
eventhandler
onFrameChanged
open
private
scroll
scrollIntoView [2]
scrollList
append
close
delete
deleteFirst
deleteLast
eventhandler
onActivate
onDeactivate
onFrameChanged
onMouseDown
onRefresh
onSelect
onVisibleChanged
select
getItemRectangle
insert
itemHeight
multipleSelection
numberOfItems
open
prepend
private
scan
scanSelection
selection
clear
deselect
first
has

    last
    scrollIntoView
    select
select [2] [3] [4]
selectAction [2]
selection [2] [3]
separator [2]
separatorStyles
set [2] [3]
setFromPoints
setFromRectangle
setOneFace
setOneFont
setOneSize
setOneStyle
setText
setupDone
shadowIn
shadowOut
shape
  eventhandler
    onHiliteChanged
    onRefresh
  fill
    backgroundColor
    foregroundColor
    tile
  open
  private
shiftKey
show [2]
showLabel
showLabelChanged
showLabelChangedAction
showModal
simple
sin_a [2]
singleDashedLine
singleLine
size [2] [3] [4] [5] [6]
specialkey
specKeys
standardEditMenu
standardFileMenu
standardMenubar
standardTextStyle
start [2] [3]
startAngle
state
stateChanged
stateChangedAction
staticText
  close
  eventhandler
    onRefresh
  open
  private
stipple [2]
stop
style [2] [3] [4] [5] [6]
styleChanged [2]
styleChangedAction [2]
styledText
  styleInfo
styleInfo
subMenu

subtract
symDiff

T

target
terminate
textChanged
textChangedAction
textContents
textEditor
  close
  contentsType
  open
  private
  scroll
textFaces
textField
  all
  clear
  close
  contents
  copy
  cut
  defaultStyle
  delete
  eventhandler
    beforeChange
    onBeforeChange
    onDisableTarget
    onEnableTarget
    onFrameChanged
    onKeyDown
    onMouseDown
    onMouseUp
    onRefresh
    onTextChanged
    textChanged
  getChar
  insert
  isOneStyle
  length
  margin
  open
  paste
  posToPt
  private
  ptToPos
  scanText
  scanTextWithStyle
  selection
    contents
    end
    get
    scrollIntoView
    set
    start
  setOneFace
  setOneFont
  setOneSize
  setOneStyle
textScrollList
  close

  getText
  open
  setText
  style
textStyle
theAction
theButton
theCursor
theCursorChanged
theCursorChangedAction
theEditMenu
theEventhandler
theFileMenu
theMenubar [2]
theMenuitem
theName
theRectangle
theta
thumbMoved
thumbMovedAction
ticks
tile
timer
title
toggleButton
  close
  eventhandler
    onMouseUp
    onStateChanged
    stateChanged
  open
  private
  state
top
topLeft
trace
trackMouse
transferModes
transformPoint
transformRectangle
transparent
transparentColor
tx
ty
type

U

undoMenuitem
union [2]
update [2]

V

v
value
valueChanged
valueChangedAction
vertical [2]
visible [2] [3]
visibleChanged

visibleChangedAction
vr

W

warnStream
watch
wedge
  endAngle
  eventhandler
    onRefresh
  open
  private
  startAngle
when
white
width
widthOfChar
widthOfText
window
windowitem
windowTypes

X

x [2]
xorBlend
XsystemEnvHandleCallback
XsystemEnvHandleCallbackP
XsystemEnvPresent

Y

y [2]

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 "size"
        *)
       (# 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 "size"
        *)
       (# 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 "size"
        *)
       (# 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 "size"
        *)
       (# 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 "size"
        *)
       (# numOfItems: @integer;
       enter numOfItems
```

```
  do ...
  #);
deleteLast:
  (* 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 "size"
   *)
  (# 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 ...; #);
  #)
```

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: ^text;
     fileDialogPrivate: @...;
  ...
  exit fileName[]
  #);
fileSelectionDialog: fileDialog
  (* brings up a standard file selection dialog *)
  (# ... #);
fileCreationDialog: fileDialog
  (* brings up a standard file creation dialog *)
  (# ... #)
```

Styledtext Interface

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