EiffelVision

Requirements Specification

ISE Technical Document

Modification date: 10/12/98

1 Introduction

1.1 Purpose

This document is the functional specification of EiffelVision GUI and graphics library. The document describes all the functional requirements for the library and function as a basis and reference when designing and programming the library.

The audience of this document are the developers and maintainers of EiffelVision. The EiffelVision programmer’s manual will be written based on this document.

1.2 Scope

This library is a new version of ISE’s EiffelVision. In this document the terms ‘EiffelVision’ and ‘the library’ refers to the new library and the term ‘old EiffelVision’ refers to the old library. Although EiffelVision is a rewrite of old EiffelVision, parts of the old EiffelVision are used in the implementation of the library whenever appropriate.

EiffelVision is a software library for application developers using Eiffel language. EiffelVision offers an object-oriented framework for both graphical user interface and graphics development.Using the library, developers have an access to all the necessary GUI components to develop a modern, functional and good-looking application. The library also offers tools to draw figures, points, lines, arcs, polygons etc., on the screen.

1.3 Definitions, Acronyms and Abbreviations

Eiffel	[Meyer 1992]
EiffelVision	GUI and graphics library for applications development described in this document.
old EiffelVision	Old library for the same purpose than the above.
GUI	Graphical User Interface.

GTK	The GIMP tookit. See section [GTK 1998].
Widget	GUI component in EiffelVision.
Developer, Library User	The application developer, who is using EiffelVision library.
User, Application User	The user of the application developed using EiffelVision

1.4 References

[Meyer 1992]

Bertrand Meyer; *Eiffel: The Language*; Prentice Hall Object-Oriented Series, 1991; second revised printing, 1992.

[GTK 1998]

GTK Web page; http://www.gtk.org/; 1998.

1.5 Overview

Chapter 2 describes the general factors that affect the library and its requirements. Chapter 3 contains the detailed requirements on the fuctions and the external interfaces of the library together with design constraints.

2

General Description

2.1 Library Perspective

EiffelVision is an abstract, is multi-platform library. Supported platforms are Microsoft Windows and Unix/X Window System using GTK toolkit [GTK 1998].

2.2 Library Functions

The main functions of the library are to provide components for building graphical user interface for an application and drawing figures onto a screen.

2.2.1 Graphical User Interface

EiffelVision provides a set of GUI components and methods to associate actions to GUI events. GUI components in EiffelVision are called widgets. Widget hierachy is presented in picture 1.

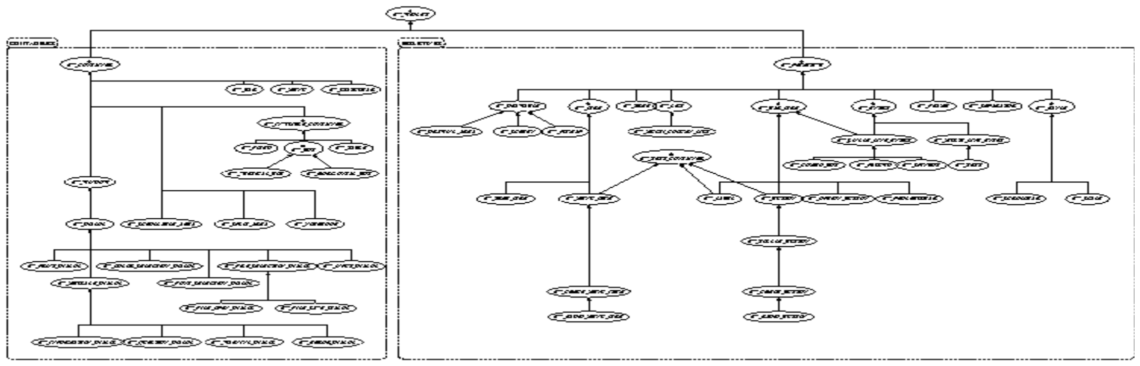


Figure 1 Class hierarchy for EiffelVision widgets

Most of the widgets in EiffelVision have events and the user of the library can associate commands to the events. For example, for a BUTTON widget there is an action **Button_ press**. In order to associate an action, for example open a dialog, to this event the user defines his own command class inheriting from a EiffelVision class COMMAND, creates an object of that type and uses a feature **add_action** from BUTTON.

2.2.2 Events

An event is an external action, usually triggered by the application's user, which may affect the execution of the application.

Simple examples of events are mouse button pushes and keystrokes. Others include timer activation, mouse movement, auto-repeating keyboard, context resize and change of window resources.

In an application, not all events will be meaningful for each context. For example a keystroke is typically ignored if it occurs outside of any window. So part of what defines an application is the two-dimensional grid of what events are meaningful for what contexts. Such a grid is called a **state domain**. Here is a simple example of state domain:

Event	Left click	Right click	Cursor out	Keyboard
Context				
Window_1			●	●
Window_2				●
Button_1	●	●	●	

The • mark signal the entries for which the given event is meaningful for the given context. For example the “Cursor out” event (which occurs if a user moves the cursor out of the current context) is meaningful for *Window_1* and *Button_1* but not for *Window_2*.

As its execution progresses, the state domain may change. For example a typical application may give you, most of the time, the choice between several possible events (left-clicking, right-clicking, moving the cursor, entering text at the keyboard) in several contexts (the windows, panels and buttons of the application), so the state domain may be quite large; but a certain operation may trigger a confirmation panel in which the *application* will only recognize two events: left-clicking on the OK button and left-clicking on the CANCEL button. This means the application has entered a new, smaller state domain.

2.2.3 Commands

When an event occurs in a certain context, and the context-event pair is part of the current state domain, the application will execute a certain action. That action is represented in Eiffel by an object — an instance of the EiffelVision class *COMMAND* (more precisely, a direct instance of one of its proper descendants).

In X toolkits such as Xt, OpenLook and Motif the closest notion is that of a callback. A callback is a reference to a certain C function; you can plant a callback in the toolkit to specify that the corresponding function must be called when a certain event occurs.

Callbacks also exist under Windows, allowing Windows components to call application-specific functions provided by an application’s developers. They make it possible to avoid the massive switch instruction that is traditionally found in Windows applications.

The EiffelVision notion of command is more abstract than the notion of callback. It conforms to the object-oriented model (every command will be an object) and provides added power: in addition to the *execute* procedure, which describes the command’s execution and corresponds to the callback, command objects may have other features, in particular a *cancel* procedure that deletes the effect of the command. This makes it possible to equip an application with an unlimited undo-redo mechanism, as described in chapter 12 of *Object-Oriented Software Construction*.

Here is a general model for a class describing undoable commands:

```
deferred class
    UNDOABLE_COMMAND

inherit

    COMMAND

feature
    undoable: BOOLEAN is True;
    execute is
            -- Execute the action of this command
        deferred
        ensure
```

```
                done: not undone
            end

        undo is
                -- Cancel the action of this command
            deferred
            ensure
                undoing_occured: undone
            end

        redo is
                -- Re-execute previously undone command
            require
                undone: undone
            deferred
            ensure
                executed: not undone
            end

    feature {NONE}

        undone: BOOLEAN
            -- Has the command been undone?

    end -- class COMMAND
```

The *redo* command is often, but not always, identical to *execute*.

A list of objects of type *UNDOABLE_COMMAND* is called a **history list**. Keeping a history list enables an application to support a multiple-level undo-redo mechanism. When a user requests an “undo”, the application can simply execute

```
history_list.item.undo;
history_list.back
```

Dynamic binding ensures that the proper version of *undo* is applied to each selected command (*history_list.item*) in the list. A similar scheme is used when the user requests a “redo”.

2.2.4 Figures

The interface of an EiffelVision application may include not just predefined contexts but also graphical figures.

The EiffelVision model figures is inspired from a familiar notion: geographical maps. The design of a map uses several levels of abstraction, illustrated on the Figure 2:

- We may view the reality behind the model (in an already abstracted form) as a set of geometrical shapes or **figures**. For a map the figures represent rivers, roads, towns and other geographical objects.
- Then, the **world** is a set of such figures.
- The **windows** are rectangular areas of the world.

- The map is a representation of a part of the world which can contain one or more windows. For example a map can have one main window devoted to country, and subsidiary windows devoted to large cities or outlying parts (as with Hawaii in maps of the USA).
- And the **device** is a physical medium on which the map is displayed. The device is usually a sheet of paper, but we may also use a computer screen. Various parts of the device will be devoted to the various windows.

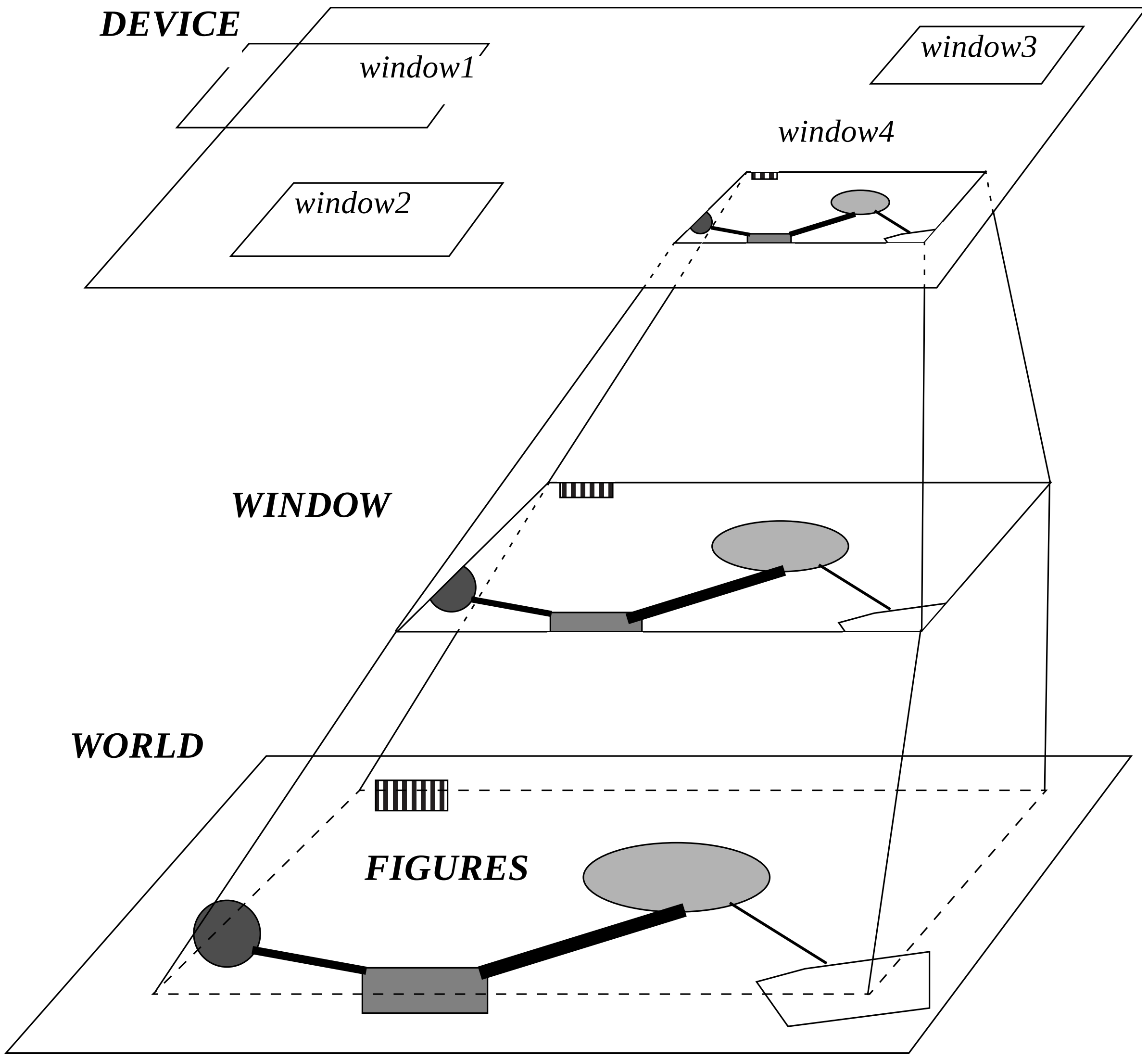


Figure 2 The graphical model

The four basic concepts — *world*, *figure*, *window*, *device* — transpose readily to general graphical applications, where the world may contain arbitrary figures of interest to a certain computer application, rather than just representations of geographical objects. Rectangular areas of the world (windows) will be displayed on rectangular areas of the device (the computer screen).

The above figure shows the three planes: world (bottom), window (middle) and device (top). The notion of window plays a central role, as each window is associated both with an area of the world and with an area of the device. Windows also cause the only significant extension to the basic map concepts: support for hierarchically nested windows. Our windows will be permitted to have subwindows, with no limit on the nesting level, although no nesting appears on the figure.

Note that two transformations are involved, both of which may include a translation and a scale factor: from world to window, and from window to device. This gives the necessary flexibility to a model, as you may:

- Move a window with respect to the world (as in when drawing a map of a different part of a country) or with respect to the device (as when moving a map on your desk).
- Change the scale of the window with respect to the world (as when changing the scale of a map, the map size remaining constant) or with respect to the device (as when deciding to use a smaller or bigger map).

2.2.5 Drag and Drop

2.2.6 Pick and Drop

2.3 User Characteristics

Users of EiffelVision are software developers who should have certain knowledge in order to use the library. The library users should have a reasonably good knowledge of Eiffel and Object-Oriented software development. They should understand the basic conceps of Graphical User Interfaces, but a platform specific knowledge of GUI programming is not necessary. For example, the library user doesn't have to know MS Windows programming, because EiffelVision is an abstract library.

3

Functional Requirements

3.1 Widgets

A widget is a basic component when building the user interface. All widgets, except for deferred widgets (marked with symbol * in picture 1.) are meant for the library user to create the corresponding GUI components.

Creating widgets is simple and consistent. All the widgets have a creation procedure make:

make (par: EV_CONTAINER) is

Procedure ‘make’ has one argument, parent, which is an EiffelVision container (See section 3.4). As all the widget need to have a parent, exept the Windows that can be create with the ‘make_top_level’ creation procedure, you need to create the parent before to create the widget.

Procedure ‘make’ creates the widget using default setting for the specific type of widget. Some widgets have additional creation routines which can be used, when more detailed control over the widgets creation is needed.

```
indexing
    description: "Most general notion of widget (i.e. user interface object)"
    status: "See notice at end of class"
    names: widget
    date: "$Date: 1998/10/02 16:58:46 $"
    revision: "$Revision: 1.19 $"

deferred class interface
    EV_WIDGET

feature -- Access
```

```
    parent: EV_WIDGET
            -- The parent of the Current widget
            -- If the widget is an EV_WINDOW without parent,
            -- this attribute will be Void
        require
            exists: not destroyed

feature -- Measurement
-- The coordinates are effective only if widget is inside a
-- fixed container. Otherwise they are calculated
-- automatically by the container widget.

    height: INTEGER
            -- Height of the widget
        require
            exists: not destroyed
        ensure
            positive_height: Result >= 0

    minimum_height: INTEGER
            -- Minimum height that application wishes widget
            -- instance to have
        require
            exists: not destroyed
        ensure
            positive_height: Result >= 0

    minimum_width: INTEGER
            -- Minimum width that application wishes widget
            -- instance to have
        require
            exists: not destroyed
        ensure
            positive_height: Result >= 0

    width: INTEGER
            -- Width of the widget
        require
            exists: not destroyed
        ensure
            positive_width: Result >= 0

    x: INTEGER
            -- Horizontal position relative to parent
        require
            exists: not destroyed;
            unmanaged: not managed

    y: INTEGER
```

```
                    -- Vertical position relative to parent
               require
                    exists: not destroyed;
                    unmanaged: not managed

     feature -- Comparison

          same (other: like Current): BOOLEAN
                    -- Does Current widget and other correspond
                    -- to the same screen object?
               require
                    other_exists: other /= void

     feature -- Status report

          automatic_position: BOOLEAN
                    -- Does the widget take a new position when
                    -- the parent resize ?
                    -- (If it does, its size doesn't changed).
                    -- False by default

          automatic_resize: BOOLEAN
                    -- Is the widget resized automatically when
                    -- the parent resize ? In this case,
                    -- automatic_position has no effect.
                    -- True by default

          destroyed: BOOLEAN
                    -- Is Current widget destroyed?
                    -- (= implementation does not exist)

          insensitive: BOOLEAN
                    -- Is current widget insensitive to
                    -- user actions?
                    -- (If it is, events will not be dispatched
                    -- to Current widget or any of its children)
               require
                    exists: not destroyed

          managed: BOOLEAN
                    -- Is the geometry of current widget managed by its
                    -- container? This is the case always unless
                    -- parent.manager = False (Always true except
                    -- when the container is EV_FIXED). This is
                    -- set in the procedure set_default

          shown: BOOLEAN
                    -- Is current widget visible?
               require
```

```
                    exists: notdestroyed

feature -- Status setting

          destroy
                    -- Destroy actual screen object of Current
                    -- widget and of all children.
               ensure
                    destroyed: destroyed

          hide
                    -- Make widget invisible on the screen.
               require
                    exists: notdestroyed
               ensure
                    not_shown: notshown

          set_automatic_position (state: BOOLEAN)
                    -- Make state the new automatic_position.
               require
                    exists: notdestroyed
               ensure
                    automatic_position_set: automatic_position = state

          set_automatic_resize (state: BOOLEAN)
                    -- Make state the new automatic_resize.
               require
                    exists: notdestroyed
               ensure
                    automatic_resize_set: automatic_resize = state

          set_insensitive (flag: BOOLEAN)
                    -- Set current widget in insensitive mode if
                    -- flag. This means that any events with an
                    -- event type of KeyPress, KeyRelease,
                    -- ButtonPress, ButtonRelease, MotionNotify,
                    -- EnterNotify, LeaveNotify, FocusIn or
                    -- FocusOut will not be dispatched to current
                    -- widget and to all its children. Set it to
                    -- sensitive mode otherwise.
               require
                    exists: notdestroyed
               ensure
                    flag = insensitive

          show
                    -- Make widget visible on the screen. (default)
               require
                    exists: notdestroyed
```

```
        ensure
            shown: shown

feature -- Resizing

    set_height (new_height: INTEGER)
            -- Make new_height the new height.
        require
            exists: notdestroyed;
            positive_height: new_height >= 0
        ensure
            dimensions_set: implementation.dimensions_set (width, new_height)

    set_minimum_height (min_height: INTEGER)
            -- Make min_height the new minimum__height.
        require
            exists: notdestroyed;
            large_enough: min_height >= 0
        ensure
            min_height = min_height

    set_minimum_size (min_width, min_height: INTEGER)
            -- Make min_width the new minimum_width
            -- and min_height the new minimum_height.
        require
            exists: notdestroyed;
            large_enough: min_height >= 0;
            large_enough: min_width >= 0
        ensure
            min_width = min_width;
            min_height = min_height

    set_minimum_width (min_width: INTEGER)
            -- Make min_width the new minimum_width.
        require
            exists: notdestroyed;
            large_enough: min_width >= 0
        ensure
            min_width = min_width

    set_size (new_width: INTEGER; new_height: INTEGER)
            -- Make new_width the new width
            -- and new_height the new height.
        require
            exists: notdestroyed;
            positive_width: new_width >= 0;
            positive_height: new_height >= 0
        ensure
            dimensions_set:   implementation.dimensions_set   (new_width,   new_
height)

        set_width (new_width: INTEGER)
                        -- Make new_width the new width.
                require
                        exists: not destroyed;
                        positive_width: new_width >= 0
                ensure
                        dimensions_set: implementation.dimensions_set (new_width, height)

        set_x (new_x: INTEGER)
                        -- Put at horizontal position new_x relative
                        -- to parent.
                require
                        exists: not destroyed;
                        unmanaged: not managed
                ensure
                        x_set: x = new_x

        set_x_y (new_x: INTEGER; new_y: INTEGER)
                        -- Put at horizontal position new_x and at
                        -- vertical position new_y relative to parent.
                require
                        exists: not destroyed;
                        unmanaged: not managed

        set_y (new_y: INTEGER)
                        -- Put at vertical position new_y relative
                        -- to parent.
                require
                        exists: not destroyed;
                        unmanaged: not managed
                ensure
                        y_set: y = new_y

feature -- Event - command association

        add_button_press_command (mouse_button: INTEGER; command: EV_COM-
MAND; arguments: EV_ARGUMENTS)
                        -- Add command to the list of commands to be executed when
                        -- button no 'mouse_button' is pressed.
                require
                        exists: not destroyed

        add_button_release_command (mouse_button: INTEGER; command: EV_COM-
MAND; arguments: EV_ARGUMENTS)
                        -- Add command to the list of commands to be executed when
                        -- button no 'mouse_button' is released.
                require
```

*exists: **not**destroyed*

add_destroy_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)

-- Add *command* to the list of commands to be executed when
-- the widget is destroyed.

require

*exists: **not**destroyed*

add_double_click_command (mouse_button: INTEGER; command: EV_COMMAND; arguments: EV_ARGUMENTS)

-- Add *command* to the list of commands to be executed when
-- button no *mouse_button* is double clicked.

require

*exists: **not**destroyed*

add_enter_notify_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)

-- Add *command* to the list of commands to be executed when
-- the cursor of the mouse enter the widget.

require

*exists: **not**destroyed*

add_expose_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)

-- Add *command* to the list of commands to be executed when
-- the widget has to be redrawn because it was exposed from
-- behind another widget.

require

*exists: **not**destroyed*

add_key_press_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)

-- Add *command* to the list of commands to be executed when
-- a key is pressed on the keyboard while the widget has the
-- focus.

require

*exists: **not**destroyed*

add_key_release_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)

-- Add *command* to the list of commands to be executed when
-- a key is released on the keyboard while the widget has the
-- focus.

require

*exists: **not**destroyed*

add_leave_notify_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)

-- Add *command* to the list of commands to be executed when

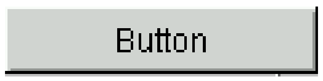
```
                -- the cursor of the mouse leave the widget.
        require
                exists: not destroyed

    add_motion_notify_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)
                -- Add command to the list of commands to be executed when
                -- mouse move.
        require
                exists: not destroyed

    last_command_id: INTEGER
                -- Id of the last command added by feature add_command
        require
                exists: not destroyed

    remove_command (command_id: INTEGER)
                -- Remove the command associated with command_id from the
                -- list of actions for this context. If there is no command
                -- associated with command_id, nothing happens.
        require
                exists: not destroyed

end -- class EV_WIDGET
```

3.2 Primitives

A primitive is a widget that has no children. It means that other widgets cannot be put inside a primitive. Some primitives can have components inside, but the type of the components is pre-defined. For example, a button can contain a pixmap component and a text component, but nothing else.

```
deferred class interface
    EV_PRIMITIVE

end -- class EV_PRIMITIVE
```

3.2.1 EV_BUTTON



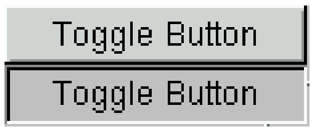
Class EV_BUTTON is one of the most useful user interface components. It is also a common ancestor for different button classes.

A button has a 3D appearance as the underlying toolkit implements it. A button can contain a text label, a pixmap, or both. When both of them are present, there are two different ways to present them; pixmap on the top and label on the bottom, or pixmap on the left and label on the right.

The reason why a button is not specified as a container is that on Windows it would be difficult to implement. It would be conceptually nicer to have button as a container and then but a label inside it when needed a button with a text and pixmap inside it when needed a button with pixmap. More complex situations would also be easy to manage. In practice, however, only labels and pixmaps are interesting as components to put inside a button.

```
indexing
        description: "EiffelVision button. Basic GUI push button. This is also abase class for
other buttons classes"
        status: "See notice at end of class"
        id: "$Id: ev_button.e,v 1.9 1998/09/22 01:46:45 samik Exp $"
        date: "$Date: 1998/09/22 01:46:45 $"
        revision: "$Revision: 1.9 $"

class interface
        EV_BUTTON

creation
        make,
        make_with_text

feature -- Access

        pixmap_container: EV_PIXMAP_CONTAINER
                        -- Pixmap inside button

feature -- Event - command association

        add_click_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)
                        -- Add ’command’ to the list of commands to be
                        -- executed when the button is pressed
                require
                        valid_command: command /= void

end -- class EV_BUTTON
```

3.2.2 EV_TOGGLE_BUTTON



EV_TOGGLE_BUTTON is a descendant of EV_BUTTON and is very similar, except that it will always be in one of two states, alternated by a click. A toggle button may be depressed, and when clicked again, it will pop back up. Click again, and it will pop back down.

Toggle buttons are the basis for check buttons and radio buttons, as such, many of the calls used for toggle buttons are inherited by radio and check buttons.

The default state after creation is ‘not pressed’.

indexing
description: "EiffelVision toggle button. It looks and acts like a button, but is always in one of two states,alternated by a click. Toggle button may bedepressed, and when clicked again, it will pop backup. Click again, and it will pop back down."
status: "See notice at end of class"
id: "$Id: ev_toggle_button.e,v 1.9 1998/09/28 16:12:26 samik Exp $"
date: "$Date: 1998/09/28 16:12:26 $"
revision: "$Revision: 1.9 $"

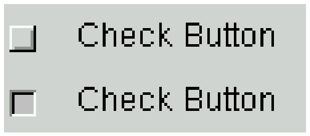
```
class interface
    EV_TOGGLE_BUTTON

creation
    make,
    make_with_text

feature -- Status report

    pressed: BOOLEAN
            -- Is toggle pressed
        require
            exists: notdestroyed

feature -- Status setting

    set_pressed (button_pressed: BOOLEAN)
            -- Set Current toggle on and set
            -- pressed to True.
        require
            exists: notdestroyed
        ensure
            correct_state: pressed = button_pressed

    toggle
            -- Change the state of the toggel button to
            -- opposite
        require
            exists: notdestroyed
        ensure
            state_is_true: pressed = notold pressed
```

feature -- Event - command association

> *add_toggle_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)*
> -- Add ’command’ to the list of commands to be
> -- executed when the button is toggled
> ***require***
> *valid_command: command /= void*

end -- class *EV_TOGGLE_BUTTON*

3.2.3 EV_CHECK_BUTTON



Check buttons are similar to toggle buttons, but they look a little different. Rather than being buttons with a label and/or a pixmap inside them, they look like check buttons on the underlying toolkit. That means usually a small square with a label right of it.

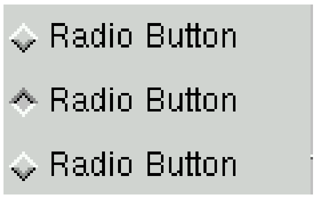
indexing
> *description: "EiffelVision Check button. Widget that has a check box and a text."*
> *status: "See notice at end of class"*
> *id: "$Id: ev_check_button.e,v 1.4 1998/09/28 16:12:24 samik Exp $"*
> *date: "$Date: 1998/09/28 16:12:24 $"*
> *revision: "$Revision: 1.4 $"*

class interface
> *EV_CHECK_BUTTON*

creation
> *make_with_text*

end -- class *EV_CHECK_BUTTON*

3.2.4 EV_RADIO_BUTTON



Radio buttons are similar to check buttons except that radio buttons are grouped and only one in a group may be selected at a time. Radio buttons having the same parent belong to the same group. Sometimes it can be possible to have several radio button groups inside the same parent, for example a window. However, this is not a problem, because special containers can be used to group radio buttons. For example, EV_FRAME is a good component to group radio buttons, because it also groups the buttons visually inside a border.

Radio buttons are good for places in an application where one option is needed to select from a short list of options.

After the creation, the radio button which was created first in the same group is pressed and the other radio buttons of the group are not pressed.

indexing
description: "EiffelVision radio button. Radio buttons aresimilar to check buttons except that radiobuttons are grouped so that only one may beselected at a time."
status: "See notice at end of class"
id: "$Id: ev_radio_button.e,v 1.3 1998/09/28 16:12:25 samik Exp $"
date: "$Date: 1998/09/28 16:12:25 $"
revision: "$Revision: 1.3 $"

class interface
EV_RADIO_BUTTON

creation
make_with_text

feature -- Initialization

make_with_text (par: EV_CONTAINER; txt: STRING)
-- radio button with *par* as parent and *txt* as
-- text label

end -- class *EV_RADIO_BUTTON*

3.2.5 EV_LABEL

A label ia a static text that can be put anywhere in a window, such as an explanation next to a text field.

indexing
description: "EiffelVision label"
status: "See notice at end of class"
id: "$Id: ev_label.e,v 1.7 1998/09/09 21:50:23 aitkaci Exp $"

date: "$Date: 1998/09/09 21:50:23 $"
revision: "$Revision: 1.7 $"

class interface
EV_LABEL

creation
make,
make_with_text

end -- class *EV_LABEL*

3.2.6 EV_TEXT_COMPONENT

EV_TEXT_COMPONENT is a deferred class and an ancestor for several classes. Later, it shouldl give several tools to manage a text. In particular, it will have features to find a string, cut, copy or past a part of the text.

indexing
description: "EiffelVision text component. Common ancestor for text classes liketext field and text area."
status: "See notice at end of class"
id: "$Id: ev_text_component.e,v 1.3 1998/10/02 17:03:50 aitkaci Exp $"
date: "$Date: 1998/10/02 17:03:50 $"
revision: "$Revision: 1.3 $"

deferred class interface
EV_TEXT_COMPONENT

feature -- Access

text: STRING
-- Text in component
require
exists: ***not****destroyed*

feature -- Status setting

append_text (txt: STRING)
-- Append *txt* into component.
require
exist: ***not****destroyed;*
not_void: txt /= void

prepend_text (txt: STRING)
-- Prepend *txt* into component.
require
exist: ***not****destroyed;*
not_void: txt /= void

```
    select_region (start_pos, end_pos: INTEGER)
                -- Select (hilight) the text between
                -- start_pos and end_pos
        require
                exist: notdestroyed;
                valid_start: start_pos > 0 and start_pos <= text.count;
                valid_end: end_pos > 0 and end_pos <= text.count

    set_maximum_line_length (length: INTEGER)
                -- Make length the new number of characters on a line.
                -- If length < text.cout then the text is truncated
        require
                exist: notdestroyed

    set_position (pos: INTEGER)
                -- Set current insertion position.
        require
                exist: notdestroyed;
                valid_pos: pos > 0 and pos <= text.count

    set_text (txt: STRING)
                -- Make txt the new text.
        require
                exist: notdestroyed;
                not_void: txt /= void
        ensure
                text_set: text.is_equal (txt)

feature -- Basic operation

    copy_selection
                -- Copy the selected_region in the Clipboard
                -- to paste it later.
                -- If the selected_region is empty, it does
                -- nothing.
        require
                exists: notdestroyed

    cut_selection
                -- Cut the selected_region by erasing it from
                -- the text and putting it in the Clipboard
                -- to paste it later.
                -- If the selectd_region is empty, it does
                -- nothing.
        require
                exists: notdestroyed

    paste (index: INTEGER)
                -- Insert the string which is in the
```

```
                  -- Clipboard at the index postion in the
                  -- text.
                  -- If the Clipboard is empty, it does nothing.
          require
                  exists: notdestroyed

      search (str: STRING): INTEGER
                  -- Search the string str in the text.
                  -- If str is find, it returns its start
                  -- index in the text, otherwise, it returns
                  -- Void
          require
                  exists: notdestroyed;
                  valid_string: str /= void

end -- class EV_TEXT_COMPONENT
```

3.2.7 EV_TEXT_FIELD

A text field allows the application user to enter one line of text. A text field has maximum length and visible length. The text can contain alphanumeric and numeric characters as well as special characters (what? Unicode? iso8851-1?), but there is no formatting for the text. All the text in the field uses the same font and color.

Sometimes it is necessary to check the validity of text inserted using the text field. For example, a text field can accept only numbers of even better, phone numbers in certain format. The following is a suggestion for the validity checking:

Offer a class EV_TEXT_FILTER with a redefineable feature *filter (char: CHARACTER): BOOLEAN*. Filter will return *True*, if the character is valid. Another creation procedure for EV_TEXT_FIELD has to be added: *make_with_filter (filter: EV_ TEXT_FILTER)*.This is not yet very effective. A better solution would be to create the filter object giving a regular expression to describe the validity of the input

```
indexing
      description: "EiffelVision text field. To query single line of text from the user"
      status: "See notice at end of class"
      id: "$Id: ev_text_field.e,v 1.2 1998/09/01 00:07:20 samik Exp $"
      date: "$Date: 1998/09/01 00:07:20 $"
      revision: "$Revision: 1.2 $"

class interface
      EV_TEXT_FIELD

creation
```

make

feature -- Event - command association

add_activate_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)
-- Add 'command' to the list of commands to be
-- executed when the text field is activated
require
valid_command: command /= void

end -- class *EV_TEXT_FIELD*

3.2.8 EV_PASSWORD_FIELD

A password field is a text field which can be used when querying a password in the application. The text typed into a password entry is not shown, but for every character typed an asterisk (*) is shown instead.

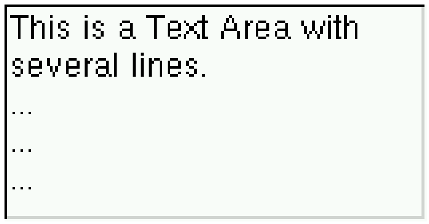
3.2.9 EV_SPINBUTTON

Spinbuttons are single line entries with two small button on the right side of the text field. The buttons have symbols arrow up and arrow down. The contents of spinbutton can only be numeric. When pressing the up button and down buttons, the value of the entry respectively increased and decreased of the chosen value.

3.2.10 EV_COMBO_BOX

A combo box contains of a text field a button. When the button is pressed, a list of possible choices is opened. Text can either be typed in to the entry field or selected from the list.

3.2.11 EV_TEXT_AREA


This is a Text Area with
several lines.
...
...
...

A text area is like a text field, but with a possibility to enter multiple lines. The property maximum length controls the number of characters on one line. When typed more characters, the cursor is automatically moved to the next line. A text area will have two creation routines, one to create a text area with or without scrollbars.

indexing

description: "EiffelVision text area. To query multiple lines of text from the user"
status: "See notice at end of class"
id: "$Id: ev_text_area.e,v 1.1 1998/08/18 01:47:04 samik Exp $"
date: "$Date: 1998/08/18 01:47:04 $"
revision: "$Revision: 1.1 $"

class interface

EV_TEXT_AREA

creation

make

end -- class *EV_TEXT_AREA*

3.2.12 EV_TEXT_EDITOR

A text editor is a complete multi-line text widget with text editing features. The text inside have several different colors and fonts.

3.2.13 EV_SEPARATOR

Separators are simple widgets that display one or several lines. They are used to separate two areas on the screen. Separators are usually used in menus, but can be used in other widgets too.

This single class specifies the direction and style of all separators. The relevant features are *set_double_dashed_line*, *set_double_line*, *set_no_line*, *set_single_dashed_line*, and *set_single_line***.** The direction can be set by *set_horizontal* and queried by *is_horizontal*.

3.2.14 EV_RANGE

EV_RANGE is a deferred class and a common ancestor for EV_SCROLLBAR and EV_SCALE.

3.2.15 EV_SCROLLBAR

A scrollbar is a simple concept. It has a thumb indicating the relative position within the scrollable material (or position within the scrollbar) and arrows at both end to give a direction indication. Usually the thumb can be dragged to move it to a specific position, clicking on the arrows moves it of one line unit and clicking near the arrows moves it of one page unit. The line and page units can be set by the user.

Scrollbars can be used by themselves to specify relative values such as sliders on a hi-fi system but are usually attached to something else.

EV_SCROLLBAR is a deferred class, it is the ancestor of EV_HORIZONTAL_SCROLLBAR and EV_VERTICAL_SCROLLBAR.

The events that can occur on a scrollbar include the movement of the thumb and the position being changed.

Depending on the toolkit, there may be a possibility to have acceleration of the speed of movement of the thumb. This is usually based on an *initial_delay* and a *repeat_delay* which can be set (*set_inital_delay* and *set_repeat_delay*). Also affecting the movement is the *granularity* which will affect how much the thumb is to move as well as the *maximum* and *minimum* of the range of movement. The routines to set the motion affecting values are *set_granularity*, *set_maximum* and *set_minimum*. The current position of the thumb can be set and queried using *set_value* and *value* respectively.

3.2.16 EV_SCALE

A scale is like a scrollbar, but is used to set or represent numeric values. It can be considered as a scrollbar with a label indicating a value. The text used for the labels in the scale is EV_FONTABLE.

EV_SCALE is a deferred class, ancestor of EV_HORIZONTAL_SCALE and EV_VERTICAL_SCALE.

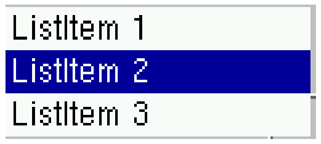
Like a scrollbar, a scale \has a move event and a value changed event. There are routines to attach and remove commands from these (*add_move_action*, *add_value_changed_action*, *remove_move_action* and *remove_value_changed_action*).

The granularity, minimum, maximum, thumb value and orientation all have the same meanings and associated routines as EV_SCROLLBAR.

The major difference between a scrollbar and the scale is the output modes of the scale. The scale may be set so that it only does output values (*set_output_only*) and have this queried (*is_output_only*). The label may be made to appear with whatever text using the *set_text* feature and queried using the *text* feature. The numerical value of the scale may be shown by setting *is_value_shown* through the *show_value* feature.

By default, the maximum of the scale is on the bottom for the vertical scales and on the right for the horizontal ones. However, this default behavior can be changed by *set_maximum_right_bottom* and can be queried with *is_maximum_right_bottom.*

3.2.17 EV_LIST



A list is a component with a list of options which may be selected by a user. There may be only one selection allowed or multiple selections allowed. The text within the list is of type EV_LIST_ITEM.

3.2.18 EV_MULTI_COLUMN_LIST

A multi column list has the functionality of list with the difference that the item in it is of type EV_MULTI_COLUMN_LIST_ITEM. A multi column list item consists of several parts so that each part is the list item represents the item in one column. A multi column list also have a title row which is displayed on top of the list. The title row controls which columns are visible and what is the visible size of the columns. Columns can be added, removed and resized.

3.2.19 EV_TREE

A tree is a component which allows data to be represented hierarchically. A single data item in a tree is of type EV_TREE_ITEM.

3.2.20 EV_MENU

A menu is a rectangular area with a vertical list of menu items. Each menu item is of type EV_MENU_ITEM.

indexing
  description: "EiffelVision menu. Menu contains menu items several menu items and shows them when the menu is opened."
  status: "See notice at end of class"
  id: "$Id: ev_menu.e,v 1.4 1998/09/11 00:53:19 samik Exp $"
  date: "$Date: 1998/09/11 00:53:19 $"
  revision: "$Revision: 1.4 $"

class interface
  EV_MENU

creation
  make_with_text

feature -- Implementation

  implementation: EV_MENU_I

end -- class *EV_MENU*

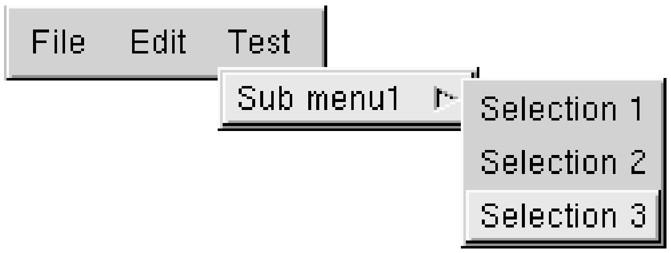
3.2.21 EV_MENU_ITEM

A menu item is a component that can be put on a menu. Menu item is shown as a piece of text.

indexing
  description: "EiffelVision menu item. Item that must be put in an EV_MENU_ITEM_CONTAINER."
  status: "See notice at end of class"

```
    id: "$Id: ev_menu_item.e,v 1.5 1998/09/22 21:40:29 aitkaci Exp $"
    date: "$Date: 1998/09/22 21:40:29 $"
    revision: "$Revision: 1.5 $"

class interface
    EV_MENU_ITEM

creation
    make_with_text

feature -- Status report

    insensitive: BOOLEAN
            -- Is current item insensitive to
            -- user actions?
        require
            exists: notdestroyed

feature -- Status setting

    set_insensitive (flag: BOOLEAN)
            -- Set current item in insensitive mode if
            -- flag.
        require
            exists: notdestroyed
        ensure
            flag = insensitive

feature -- Implementation

    implementation: EV_MENU_ITEM_I

end -- class EV_MENU_ITEM
```

3.2.22 EV_MENU_BAR



A menu bar is a group of menu-bar items that appears on the top of a window. Menus. combo-box, text fields are menu-bar items.

indexing

description: "EiffelVision menu bar. Menu bar is a vertical the screen or in the window containing menu items."

status: "See notice at end of class"

id: "$Id: ev_menu_bar.e,v 1.3 1998/09/29 02:01:21 aitkaci Exp $"

date: "$Date: 1998/09/29 02:01:21 $"

revision: "$Revision: 1.3 $"

class interface

EV_MENU_BAR

creation

make

end -- class *EV_MENU_BAR*

3.2.23 EV_OPTION_MENU

An option menu looks like a button. When it is clicked a menu of choices is opened. The user can select a choice in the menu. The selected item is shown as a label of the option menu button. On Windows there is no native option menu component, but a read-only combo box can be used instead.

3.2.24 EV_FRAME

A frame is simple widget that draws a border around its children.

3.2.25 EV_PROGRESSBAR

A progressbar can be used to show progress in the application, for example, to show the progress in compilation.

3.3 Drawables

3.3.1 EV_DRAWABLE

A drawable is a common ancestor for component that can contain pictures. These pictures can be pixmaps or drawn using figures (see section 3.9).

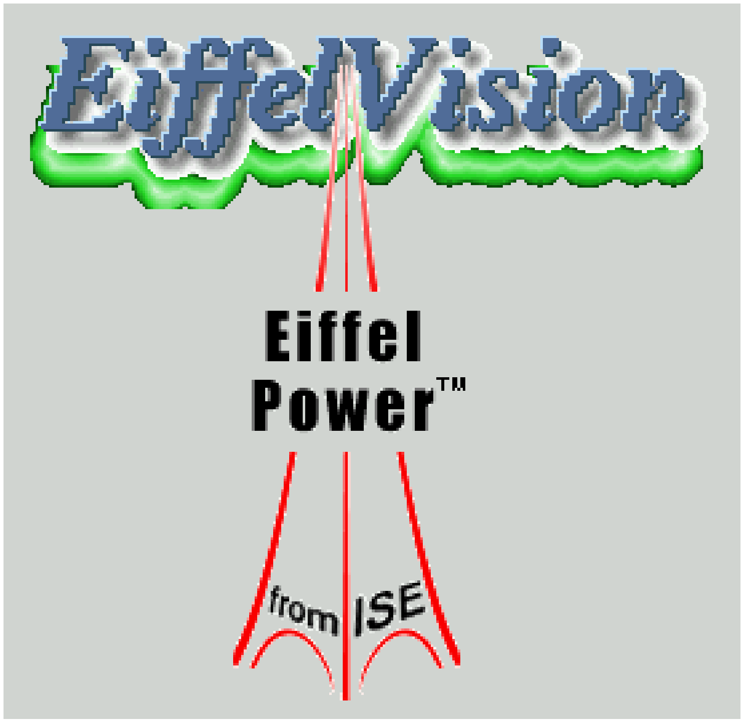
3.3.2 EV_SCREEN

A screen is a drawable and refers to the screen outside the applications windows. By using the class EV_SCREEN the application can draw figures and pixmaps anywhere on the screen without even having to open any windows.

3.3.3 EV_DRAWING_AREA

Drawing area is a widget that can contain pictures.

3.3.4 EV_PIXMAP



A pixmap is a picture consisting of several pixels of possibly different colors (pixmap = pixel map). The current implementation of pixmap is a pixmap widget, but pixmap should exist as a separate structure. The pixmap widget should be removed completely and used drawable with a pixmap component instead.

Pixmap is itself a drawable, but it can be put inside of any drawable.

indexing

description: "EiffelVision pixmap. Pixmap is a data structure that contains a picture."
status: "See notice at end of class"
id: "$Id: ev_pixmap.e,v 1.3 1998/09/17 22:59:47 samik Exp $"
date: "$Date: 1998/09/17 22:59:47 $"
revision: "$Revision: 1.3 $"

class interface

EV_PIXMAP

creation

make,
make_from_file

```
feature -- Element change

        read_from_file (file_name: STRING)
                        -- Load the pixmap described in 'file_name'.
                        -- If the file does not exist, an exception is
                        -- raised.
                        -- What about a file in wrong format?
                require
                        file_name_exists: file_name /= void

end -- class EV_PIXMAP
```

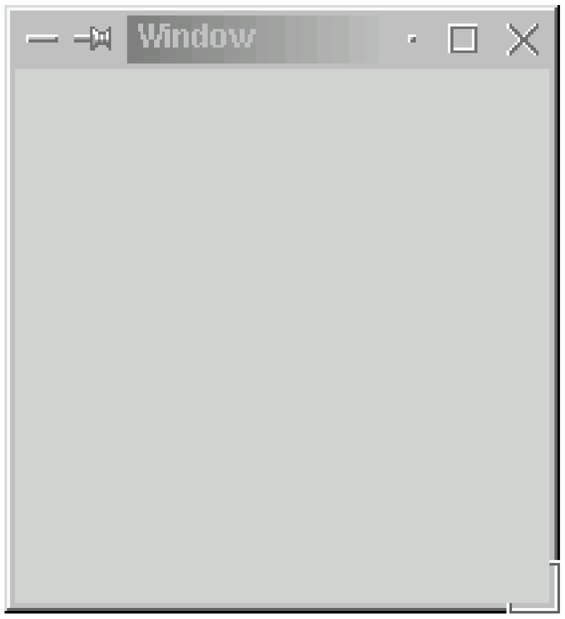
3.4 Containers

A container is a widget which allows other widgets, called its ‘children’, to be put inside it. Some of the containers allow only one child. However, because the child can also be a container, it is possible to put several widgets inside any container. See the discussion about containers fixed, box, etc...

Usually container manages its children. It means that the size and position of a child are specified by the container. The child can only specify its size and location under the restrictions of the container. For example, child can set the minimum size, but not the actual size. Also the attributes *automatic_position* and *automatic_resize* of EV_ WIDGET control the appearance of the child inside a container. The only container which does not manage its child is fixed container.

```
indexing
        description: "EiffelVision container. Container is a widget that can hold children in-
side it"
        status: "See notice at end of class"
        id: "$Id: ev_container.e,v 1.6 1998/09/29 02:01:18 aitkaci Exp $"
        date: "$Date: 1998/09/29 02:01:18 $"
        revision: "$Revision: 1.6 $"

deferred class interface
        EV_CONTAINER

feature -- Access

        client_height: INTEGER
                        -- Height of the client area (area of the
                        -- widget excluding the borders etc) of
                        -- container
                require
                        exists: notdestroyed
                ensure
                        positive_result: Result >= 0

        client_width: INTEGER
                        -- Width of the client area (area of the
                    -- widget excluding the borders etc) of
                    -- container
          require
                    exists: not destroyed
          ensure
                    positive_result: Result >= 0

     manager: BOOLEAN
                    -- Manager container manages the geometry of its
                    -- child(ren). Default True.

end -- class EV_CONTAINER
```

3.4.1 EV_WINDOW



A window is a bordered rectangular area visible on the screen. A window is a basic GUI component and a basis for almost every application. A window is a container and any widget, except for a window, can be put inside it. A window also has properties menubar, toolbar and statusbar. All of them can be visible on non visible. A menubar is the topmost component in the window, just below the window borders. A toolbar is located just below the menubar. A statusbar is the component on the bottom of the window. Should we allow floating menu- or toolbars?. The class interface is presented below.

```
indexing
     description: "EiffelVision window. Window is a visible window on the screen."
     status: "See notice at end of class"
     id: "$Id: ev_window.e,v 1.12 1998/10/02 17:02:04 aitkaci Exp $"
     date: "$Date: 1998/10/02 17:02:04 $"
     revision: "$Revision: 1.12 $"
```

```
class interface
        EV_WINDOW

creation
        make,
        make_top_level

feature -- Access

        icon_mask: EV_PIXMAP
                        -- Bitmap that could be used by window manager
                        -- to clip icon_pixmap bitmap to make the
                        -- icon nonrectangular
                require
                        exists: notdestroyed

        icon_name: STRING
                        -- Short form of application name to be
                        -- displayed by the window manager when
                        -- application is iconified
                require
                        exists: notdestroyed

        icon_pixmap: EV_PIXMAP
                        -- Bitmap that could be used by the window manager
                        -- as the application’s icon
                require
                        exists: notdestroyed
                ensure
                        valid_result: Result /= void

        parent: EV_WINDOW
                        -- The parent of the Current window: a window.
                        -- If the window is a top level, this attribute
                        -- is Void.
                        -- (from EV_WIDGET)

        title: STRING
                        -- Application name to be displayed by
                        -- the window manager
                require
                        exists: notdestroyed

        widget_group: EV_WIDGET
                        -- Widget with wich current widget is associated.
                        -- By convention this widget is the "leader" of a group
                        -- widgets. Window manager will treat all widgets in
                        -- a group in some way; for example, it may move or
                -- iconify them together
            require
                exists: notdestroyed

feature -- Measurement

    maximum_height: INTEGER
                -- Maximum height that application wishes widget
                -- instance to have
            require
                exists: notdestroyed
            ensure
                Result >= 0

    maximum_width: INTEGER
                -- Maximum width that application wishes widget
                -- instance to have
            require
                exists: notdestroyed
            ensure
                Result >= 0

feature -- Status report

    is_iconic_state: BOOLEAN
                -- Does application start in iconic state?
            require
                exists: notdestroyed

feature -- Status setting

    set_iconic_state
                -- Set start state of the application
                -- to be iconic.
            require
                exists: notdestroyed

    set_maximize_state
                -- Set start state of the application to be
                -- maximized.
            require
                exists: notdestroyed

    set_normal_state
                -- Set start state of the application to be normal.
            require
                exists: notdestroyed

feature -- Element change
```

```
        set_close_command (c: EV_COMMAND)

        set_icon_mask (mask: EV_PIXMAP)
                        -- Set icon_mask to mask.
                require
                        exists: notdestroyed;
                        not_mask_void: mask /= void

        set_icon_name (new_name: STRING)
                        -- Set icon_name to new_name.
                require
                        exists: notdestroyed;
                        valid_name: new_name /= void

        set_icon_pixmap (pixmap: EV_PIXMAP)
                        -- Set icon_pixmap to pixmap.
                require
                        exists: notdestroyed;
                        not_pixmap_void: pixmap /= void

        set_title (new_title: STRING)
                        -- Set title to new_title.
                require
                        exists: notdestroyed;
                        not_title_void: new_title /= void

        set_widget_group (group_widget: EV_WIDGET)
                        -- Set widget_group to group_widget.
                require
                        exists: notdestroyed

feature -- Resizing

        set_maximum_height (max_height: INTEGER)
                        -- Make max_height the new maximum_height.
                require
                        exists: notdestroyed;
                        large_enough: max_height >= 0
                ensure
                        max_height = max_height

        set_maximum_width (max_width: INTEGER)
                        -- Make max_width the new maximum_width.
                require
                        exists: notdestroyed;
                        large_enough: max_width >= 0
                ensure
                        max_width = max_width
```

end -- class *EV_WINDOW*

3.4.2 EV_DIALOG

Dialog is a special window which can be used for pop-up messages to the user, and other similar tasks.

3.4.3 EV_PRINT_DIALOG

3.4.4 EV_COLOR_SELECTION_DIALOG

3.4.5 EV_FONT_SELECTION_DIALOG

3.4.6 EV_FILE_SELECTION_DIALOG

3.4.7 EV_FILE_OPEN_DIALOG

3.4.8 EV_FILE_SAVE_DIALOG

3.4.9 EV_INPUT_DIALOG

3.4.10 EV_MESSAGE_DIALOG

3.4.11 EV_INFORMATION_DIALOG

3.4.12 EV_QUESTION_DIALOG

3.4.13 EV_WARNING_DIALOG

3.4.14 EV_ERROR_DIALOG

3.4.15 EV_FIXED

Fixed is an invisible container that allows unlimited number of other widgets to be put inside it. The location of widgets inside a fixed widget is specified by coordinates relative to the top left corner of fixed. The coordinates are widget attributes *x* and *y*. Fixed is the only container that allow the children specify their location and size freely.

indexing

description: "EiffelVision fixed. Invisible container that allows unlimited number of other widgets to be put inside it. The location of each widget inside is specified by the coordinates of the widget."

status: "See notice at end of class"

id: "$Id: ev_fixed.e,v 1.5 1998/09/29 02:01:18 aitkaci Exp $"

date: "$Date: 1998/09/29 02:01:18 $"

revision: "$Revision: 1.5 $"

class interface
EV_FIXED

creation
make

feature -- Access

manager: BOOLEAN

end -- class *EV_FIXED*

3.4.16 EV_BOX

Box, like fixed, is meant to be used to collect other widgets and control their appearance. Using box, widgets can be packed horizontally or vertically. Box controls the position of the widgets inside it and it can do automatic resizing. Widget inside a box can be used to right justified or left justified. EV_BOX is a deferred class, with effective descendants horizontal box and vertical box.

By default a box is homogeneous, which means that the space for all the children are is be the same size than the space for the largest child. Children can be resized to fill the space of to be in the center of the space (controlled by widget's attributes *automatic_resize* and *automatic_position*). Box can be set to non homogeneous by using the feature *set_homegeneous* with a parameter *False*. If the box is non homogeneous, each child has a space relative to the size of the child itself.

The default spacing between the children is 0. That can be changed by the feature *set_spacing*.

indexing
description: "EiffelVision box. Invisible container that allows unlimited number of other widgets to be packed inside it. Box controls the location the children%'s location and size automatically."
status: "See notice at end of class"
id: "$Id: ev_box.e,v 1.8 1998/09/29 02:01:17 aitkaci Exp $"
date: "$Date: 1998/09/29 02:01:17 $"
revision: "$Revision: 1.8 $"

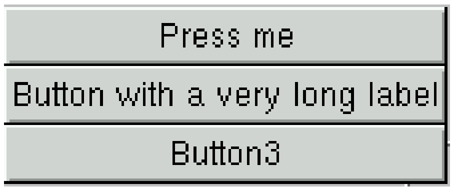
deferred class interface
EV_BOX

feature -- Element change (box specific)

set_homogeneous (homogeneous: BOOLEAN)
-- Homogenous controls whether each object in
-- the box has the same size. If homogenous =
-- True, expand argument for each child is

```
                    -- automatically True
          require
                    exist: not destroyed

     set_spacing (spacing: INTEGER)
                    -- Spacing between the objects in the box
          require
                    exist: not destroyed

end -- class EV_BOX
```

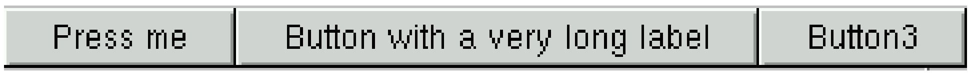
3.4.17 EV_VERTICAL_BOX



A box in vertical position.

```
indexing
     description: "EiffelVision vertical box."
     status: "See notice at end of class"
     id: "$Id: ev_vertical_box.e,v 1.4 1998/09/29 02:01:22 aitkaci Exp $"
     date: "$Date: 1998/09/29 02:01:22 $"
     revision: "$Revision: 1.4 $"

class interface
     EV_VERTICAL_BOX

creation
     make

end -- class EV_VERTICAL_BOX
```

3.4.18 EV_HORIZONTAL_BOX



A box in horizontal position.

indexing

description: "EiffelVision horizontal box."
status: "See notice at end of class"
id: "$Id: ev_horizontal_box.e,v 1.4 1998/09/29 02:01:19 aitkaci Exp $"
date: "$Date: 1998/09/29 02:01:19 $"
revision: "$Revision: 1.4 $"

class interface

EV_HORIZONTAL_BOX

creation

make

end -- class *EV_HORIZONTAL_BOX*

3.4.19 EV_TABLE

Tables are another way to pack widgets. Table contains a grid of rows and columns where the widgets are placed in. The widgets may take up as many spaces in the table as specified.

The homogeneous attribute of the table has to do with how the table's boxes are sized. If homogeneous is *True*, the table boxes are resized to the size of the largest widget in the table. If homogeneous is *False*, the size of a table boxes is dictated by the tallest widget in its same row, and the widest widget in its column.

The rows and columns are laid out from 0 to n, where n is the last row or column. A table layout with two rows and two columns is presented in Figure 3.4.20.

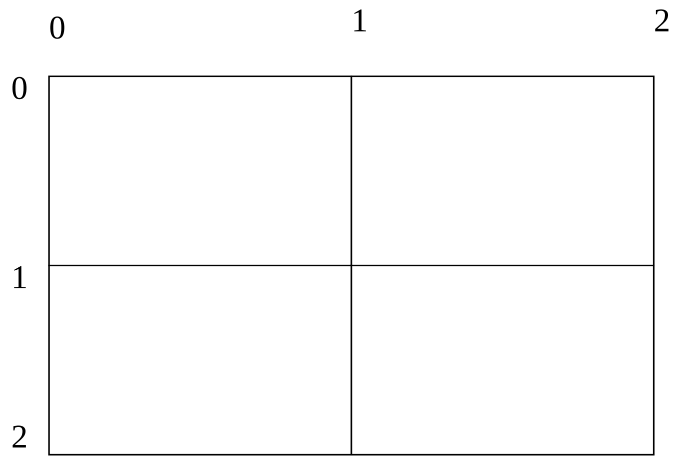
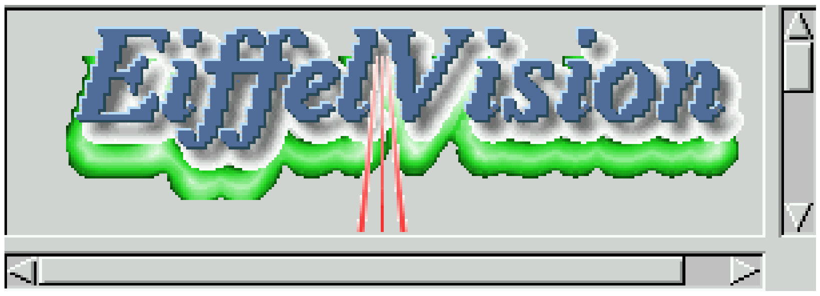


Figure 3.4.20 Table layout with two rows and two columns

The coordinate system starts in the upper left hand corner.

3.4.21 EV_SCROLLABLE_AREA



Scrollable area is a container widget with horizontal and vertical scrollbars around it. Any widget, except for a window, can be put inside a scrollable area the scrollable area offers automatic scrolling. If the widget inside is bigger than the visible size of scrollable area, the scrollbars can be used to move the view of the widget. Size of the thumbs of the scrollbar corresponds to the visible size of the widget (the size of the scrollable area) and the size of the whole scrollbar corresponds to the size of the whole widget.

indexing

description: "EiffelVision scrollable area. Scrollable area is a container with scroll-bars. Scrollable area offers automatic scrolling for its child."
status: "See notice at end of class"
id: "$Id: ev_scrollable_area.e,v 1.2 1998/09/11 19:53:11 samik Exp $"
date: "$Date: 1998/09/11 19:53:11 $"
revision: "$Revision: 1.2 $"

class interface

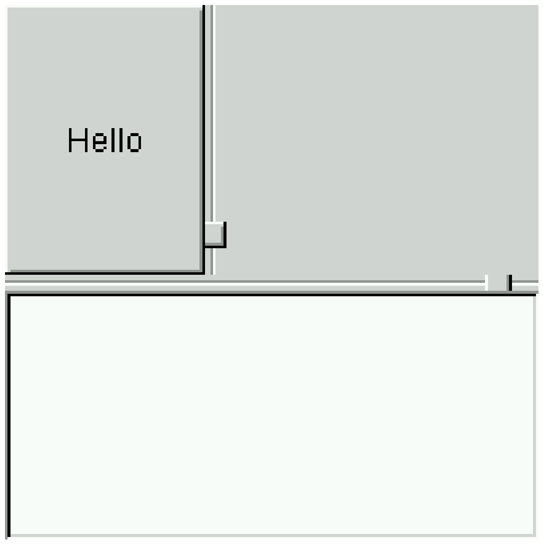
EV_SCROLLABLE_AREA

creation

make

end -- class *EV_SCROLLABLE_AREA*

3.4.22 EV_SPLIT_AREA



Split area is a container widget with two children with groove drawn between them. The user can control the relative size of the two parts by moving the groove. Split area can be either horizontal of vertical.

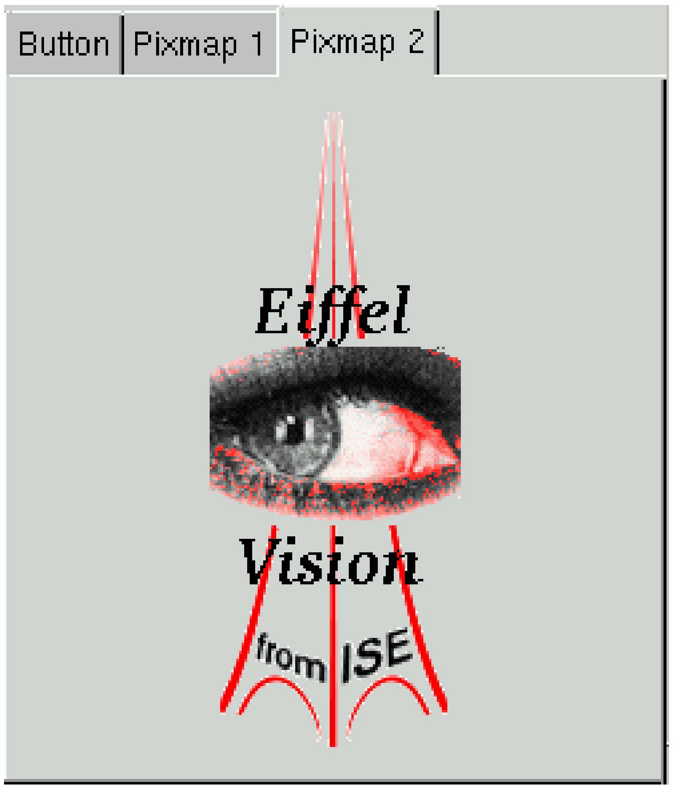
indexing

description: "EiffelVision split area. Split consists of two parts divided by a groove, which can be moved by the user to change the visible portion of the parts. Split is an abstract class with effective decendants horizontal and vertical split."
status: "See notice at end of class"
id: "$Id: ev_split_area.e,v 1.3 1998/09/29 02:01:22 aitkaci Exp $"
date: "$Date: 1998/09/29 02:01:22 $"
revision: "$Revision: 1.3 $"

deferred class interface
EV_SPLIT_AREA

end -- class *EV_SPLIT_AREA*

3.4.23 EV_NOTEBOOK



Notebook is a collection of pages that overlap each other. For each page there is a tab corresponding to the page. Only one of the pages is visible, but the tabs are visible either top, bottom, left or right of the page. When the tab is clicked, the corresponding page is made visible. If there is a lot of tabs, it is usually not possible to show them all at the same time. The number of visible tabs can be set. If there is more tabs than visible tabs, scroll buttons are shown and they can be used to control which of the tabs are visible.

Each page is a container that allows one widget to be put inside it. Pages can be added to and removed from the notebook.

indexing

description: "EiffelVision notebook. Notebook is a collection of pages that overlap each other. For each page there is a tab corresponding to the page."
status: "See notice at end of class"
id: "$Id: ev_notebook.e,v 1.3 1998/08/08 21:08:20 samik Exp $"
date: "$Date: 1998/08/08 21:08:20 $"
revision: "$Revision: 1.3 $"

class interface

EV_NOTEBOOK

creation

make

feature -- Status setting

```
    set_tab_left
                    -- set position of tabs to left
            require
                    exists: not destroyed

feature -- Element change

    append_page (c: EV_WIDGET; label: STRING)
                    -- New page for notebook containing child 'c' with tab
                    -- label 'label
            require
                    exists: not destroyed;
                    child_of_notebook: c.parent = Current

end -- class EV_NOTEBOOK
```

3.5 Events

EiffelVision has general events, which are common for all the widgets, and widget specific events.

3.5.1 General Events

The following list describes the general events.

button press	A mouse button is pressed over the widget.
button release	A mouse button is released over the widget.
double click	a mouse button is double clicked over the widget.
motion notify	Mouse pointer is moved over the widget.
delete	The widget is deleted.
expose	A part of the widget has to be redrawn because it was exposed from behind another widget.
key press	A key is pressed over the widget.
key release	A key is released over the widget.
enter notify	Mouse pointer is enters the area of widget.
leave notify	Mouse pointer leaves the area of widget.
more??	

3.5.2 Widget Specific Events

As the name suggests these events are specific for each widget. For example, button has a click event which happens when a button widget is clicked. For information on events specific to a widget, see the definition of widget in section 3.1.

3.6 Commands

A command is an object created by the library user to perform some action in response to an event. EiffelVision offers a deferred class EV_COMMAND. The library user can inherit from EV_COMMAND, to define a new command class and redefine feature *execute*. The routine *execute* is executed in response to an event.

```
indexing
        description: "General notion of command (semantic unity). To write an actual com-
mand inherit from this class and implement the ‘execute%’ feature"
        status: "See notice at end of class"
        date: "$Date: 1998/08/28 01:16:17 $"
        revision: "$Revision: 1.2 $"

deferred class interface
        EV_COMMAND

feature -- Access

        event_data: EV_EVENT_DATA
                        -- Information related to Current command,
                        -- provided by the underlying user interface
                        -- mechanism

feature -- Status report
--XX check the purpose of this this

        event_data_useful: BOOLEAN
                        -- Should the context data be available
                        -- when Current command is invoked as a
                        -- callback

        is_template: BOOLEAN
                        -- Is the current command a template, in other words,
                        -- should it be cloned before execution?
                        -- If true, EiffelVision will clone Current command
                        -- whenever it is invoked as a callback

feature -- Basic operations

        execute (arguments: EV_ARGUMENTS)
                        -- Execute Current command.
                        -- arguments is automatically passed by
```

-- EiffelVision when Current command is
-- invoked as a callback.

execute_address: POINTER
-- Address of feature execute

end -- class *EV_COMMAND*

The above specification and class interface are only temporary. The final implementation will use routine objects when they are available. It means that the library user doesn't have to create a new class for each command by inheriting from EV_ COMMAND.Instead there are two choices for EiffelVision: one approach would be to have a class EV_COMMAND with a creation routine taking a routine object and the user argument (a tuple) as arguments. Then the execution function of this command would execute the routine abject with the argument and the event_data as parameters. The other choice is just to put the routine object as an argument for *add_command* features.

3.7 Arguments

Arguments for commands are currently passed using an object of class EV_ ARGUMENTS and its descendants. However, the final implementation will use tuples instead, when available.

Thanks to the tuple mechanism, argument types are checked at compile time, so that the argument passing mechanism is type-safe.

3.8 Event Data

Event data is information specific to an event, for example, the location of the mouse pointer. Widget specific events do not have any event data.

indexing
description: "EiffelVision event data. Information given byEiffelVision when a call-back is triggered.This is the base class for representing event data"
status: "See notice at end of class"
id: "$Id: ev_event_data.e,v 1.7 1998/09/29 02:01:16 aitkaci Exp $"
date: "$Date: 1998/09/29 02:01:16 $"
revision: "$Revision: 1.7 $"

class interface
EV_EVENT_DATA

creation
make

feature -- Access

widget: EV_WIDGET
-- The mouse pointer was over this widget
-- when event happened

feature -- Debug

print_contents

end -- class *EV_EVENT_DATA*

3.8.1 EV_BUTTON_EVENT_DATA

This class represents event data for button events: button press, button release and double click.

indexing
description: "EiffelVision button event data.Class for representing button event data"
status: "See notice at end of class"
id: "$Id: ev_button_event_data.e,v 1.6 1998/08/28 00:44:12 samik Exp $"
date: "$Date: 1998/08/28 00:44:12 $"
revision: "$Revision: 1.6 $"

class interface
EV_BUTTON_EVENT_DATA

creation
make

feature -- Access

button: INTEGER

keyval: INTEGER

state: INTEGER

x: DOUBLE
-- x coordinate of mouse pointer

y: DOUBLE
-- y coordinate of mouse pointer

feature -- Debug

print_contents
-- print the contents of the object

end -- class *EV_BUTTON_EVENT_DATA*

3.8.2 EV_MOTION_EVENT_DATA

indexing

description: "EiffelVision motion event data.Class for representing motion event data"
status: "See notice at end of class"
id: "$Id: ev_motion_event_data.e,v 1.5 1998/09/03 23:32:20 samik Exp $"
date: "$Date: 1998/09/03 23:32:20 $"
revision: "$Revision: 1.5 $"

class interface
EV_MOTION_EVENT_DATA

creation
make

feature -- Initialization

make

feature -- Access

state: INTEGER

x: DOUBLE
-- x coordinate of mouse pointer

y: DOUBLE
-- y coordinate of mouse pointer

feature -- Debug

print_contents
-- print the contents of the object

end -- class *EV_MOTION_EVENT_DATA*

3.8.3 EV_KEY_EVENT_DATA

indexing

description: "EiffelVision key event data.Class for representing button event data"
status: "See notice at end of class"
id: "$Id: ev_key_event_data.e,v 1.1 1998/08/28 00:44:15 samik Exp $"
date: "$Date: 1998/08/28 00:44:15 $"
revision: "$Revision: 1.1 $"

class interface
EV_KEY_EVENT_DATA

```
creation
        make

feature -- Access

        keyval: INTEGER

        length: INTEGER

        state: INTEGER

        string: STRING

feature -- Debug

        print_contents
                        -- print the contents of the object

end -- class EV_KEY_EVENT_DATA
```

3.9 Figures

Figures work as in old EiffelVision and the implementation will be taken from there as far as possible.

3.10 Timers?

3.11 Other notes

Colormap handling under X Windows System so that it always gets the closest color.

3.12 Using the Library

EiffelVision has been designed to be simple to use and still effective. This section gives examples of using the library.

The following class implements a main window of an EiffelVision example. The main window consist of button box with toggle buttons containing label and text. Each button is associated to a command, which opens a demo window.

```
indexing
        description: "MAIND_WINDOW, main window for the application. Belongs to
EiffelVision example.";
        status: "See notice at end of class";
        id: "$Id: main_window.e,v 1.18 1998/09/23 00:11:28 samik Exp $";
        date: "$Date: 1998/09/23 00:11:28 $";
        revision: "$Revision: 1.18 $"
```

```
class MAIN_WINDOW

inherit
        EV_WINDOW
                redefine
                        make_top_level
                end;
        EV_COMMAND

creation
        make_top_level

feature --Access

        container: EV_VERTICAL_BOX;
                        -- Push buttons

        current_demo_window: DEMO_WINDOW;

feature -- Initialization

        make_top_level is
                local
                        b: MAIN_WINDOW_BUTTON;
                        c1: LABEL_DEMO_WINDOW;
                        c2: FIXED_DEMO_WINDOW;
                        c3: BOX_DEMO_WINDOW;
                        c4: NOTEBOOK_DEMO_WINDOW;
                        c5: TEXT_FIELD_DEMO_WINDOW;
                        c6: TEXT_AREA_DEMO_WINDOW;
                        c7: MENU_DEMO_WINDOW;
                        c8: SPLIT_AREA_DEMO_WINDOW;
                        c9: SCROLLABLE_AREA_DEMO_WINDOW;
                        c10: BUTTONS_DEMO_WINDOW
                do
                         precursor;
                        !! container.make (Current);
                        !! c1.make (Current);
                        !! c2.make (Current);
                        !! c3.make (Current);
                        !! c4.make (Current);
                        !! c5.make (Current);
                        !! c6.make (Current);
                        !! c7.make (Current);
                        !! c8.make (Current);
                        !! c9.make (Current);
                        !! c10.make (Current);
                        !! b.make_button (Current, "Label", "", c1);
                        !! b.make_button (Current, "Buttons", "../pixmaps/buttons.xpm", c10);
                        !! b.make_button (Current, "Fixed", "../pixmaps/fixed.xpm", c2);
                        !! b.make_button (Current, "Box", "../pixmaps/box.xpm", c3);
                        !! b.make_button (Current, "Notebook", "../pixmaps/notebook.xpm",
c4);
                        !! b.make_button (Current, "Text field", "../pixmaps/text_field.xpm",
c5);
                        !! b.make_button (Current, "Text area", "../pixmaps/text_area.xpm",
c6);
                        !! b.make_button (Current, "Menu", "../pixmaps/menu.xpm", c7);
                        !! b.make_button (Current, "Split area", "../pixmaps/split_area.xpm",
c8);
                        !! b.make_button (Current, "Scrollable area", "../pixmaps/scrollable_
area.xpm", c9);
                        set_values
                end;

feature -- Status setting

        execute (arg: EV_ARGUMENT1 [DEMO_WINDOW]) is
                        -- called when actions window is deleted
                do
                        arg.first.effective_button.set_pressed (false)
                        arg.first.actions_window.destroy
                        set_insensitive (false)
                end;

feature -- Status setting

        set_values is
                do
                        set_title ("Test all widgets")
                end;

end -- class MAIN_WINDOW
```

The following class presents the main window button used in the code example of main window. Together these example show how to create a complete user interface easily. The interface is built without specifying any coordinates nor sizes for the widgets. Everything is calculated automatically at run time. The example will be even simpler when tuples and routine objects are available and used in command and argument implementation.

```
indexing
        description: "main window button for the application. Belongs to EiffelVision
example.";
        status: "See notice at end of class";
        id: "$Id: main_window_button.e,v 1.5 1998/09/22 22:32:34 samik Exp $";
        date: "$Date: 1998/09/22 22:32:34 $";
        revision: "$Revision: 1.5 $"
```

```
class MAIN_WINDOW_BUTTON

creation
        make_button

feature {NONE} -- Initialization

        initialize (par: EV_CONTAINER) is
                        -- Common initialization for buttons
                        -- (from EV_BUTTON)
                do
                        widget_make (par)
                        !! pixmap_container.make_from_primitive (Current)
                end;

        make (par: EV_CONTAINER) is
                        -- Empty button
                        -- (from EV_TOGGLE_BUTTON)
                do
                        !EV_TOGGLE_BUTTON_IMP! implementation.make (par)
                        initialize (par)
                end;

        make_with_text (par: EV_CONTAINER; txt: STRING) is
                        -- Button with 'par' as parent and 'txt' as
                        -- text label
                        -- (from EV_TOGGLE_BUTTON)
                do
                        !EV_TOGGLE_BUTTON_IMP! implementation.make_with_text (par,
txt)
                        initialize (par)
                end;

        widget_make (par: EV_CONTAINER) is
                        -- Create a widget with par as parent and
                        -- call set_default.
                        -- This is a general initialization for
                        -- widgets and has to be called by all the
                        -- widgets with parents.
                        -- (from EV_WIDGET)
                require -- from EV_WIDGET
                        valid_parent: par /= void
                do
                        parent := par
                        set_default
                ensure -- from EV_WIDGET
                        parent_set: parent.child = Current and par = parent;
                        exists: notdestroyed
                end;
```

```
feature {NONE} --Initialization

    make_button (main_w: MAIN_WINDOW; button_name, pixmap_file_name: STRING;
cmd: DEMO_WINDOW) is
        local
            p: EV_PIXMAP;
            a: EV_ARGUMENT2 [MAIN_WINDOW, EV_TOGGLE_BUTTON]
        do
            make (main_w.container);
            set_text (button_name);
            if pixmap_file_name /= void and then notpixmap_file_name.empty
then
                !! p.make_from_file (pixmap_container, pixmap_file_name)
            end;
            !! a.make_2 (main_w, Current);
            add_toggle_command (cmd, a)
        end;

feature -- Access

    font: EV_FONT is
            -- Font name of label
            -- (from EV_FONTABLE)
        require -- from EV_FONTABLE
            exists: notdestroyed
        do
            Result := implementation.font
        end;

    parent: EV_CONTAINER;
            -- Parent container of this widget
            -- (from EV_WIDGET)

    pixmap_container: EV_PIXMAP_CONTAINER;
            -- Pixmap inside button
            -- (from EV_BUTTON)

    text: STRING is
            -- Text of current label
            -- (from EV_TEXT_CONTAINER)
        require -- from EV_TEXT_CONTAINER
            exists: notdestroyed
        do
            Result := implementation.text
        end;

feature -- Measurement
-- The coordinates are effective only if widget is inside a
-- fixed container. Otherwise they are calculated
-- automatically by the container widget.

        height: INTEGER is
                        -- Height of widget
                        -- (from EV_WIDGET)
                require -- from EV_WIDGET
                        exists: notdestroyed
                do
                        Result := implementation.height
                ensure -- from EV_WIDGET
                        positive_height: Result >= 0
                end;

        maximum_height: INTEGER is
                        -- Maximum height that application wishes widget
                        -- instance to have
                        -- (from EV_WIDGET)
                require -- from EV_WIDGET
                        exists: notdestroyed
                do
                        Result := implementation.maximum_height
                ensure -- from EV_WIDGET
                        Result >= 0
                end;

        maximum_width: INTEGER is
                        -- Maximum width that application wishes widget
                        -- instance to have
                        -- (from EV_WIDGET)
                require -- from EV_WIDGET
                        exists: notdestroyed
                do
                        Result := implementation.maximum_width
                ensure -- from EV_WIDGET
                        Result >= 0
                end;

        minimum_height: INTEGER is
                        -- Minimum height that application wishes widget
                        -- instance to have
                        -- (from EV_WIDGET)
                require -- from EV_WIDGET
                        exists: notdestroyed
                do
                        Result := implementation.minimum_height
                ensure -- from EV_WIDGET
                        positive_height: Result >= 0
                end;
```

```
    minimum_width: INTEGER is
                -- Minimum width that application wishes widget
                -- instance to have
                -- (from EV_WIDGET)
        require -- from EV_WIDGET
                exists: notdestroyed
        do
                Result := implementation.minimum_width
        ensure -- from EV_WIDGET
                positive_height: Result >= 0
        end;

    width: INTEGER is
                -- Width of widget
                -- (from EV_WIDGET)
        require -- from EV_WIDGET
                exists: notdestroyed
        do
                Result := implementation.width
        ensure -- from EV_WIDGET
                positive_width: Result >= 0
        end;

    x: INTEGER is
                -- Horizontal position relative to parent
                -- (from EV_WIDGET)
        require -- from EV_WIDGET
                exists: notdestroyed;
                unmanaged: notmanaged
        do
                Result := implementation.x
        end;

    y: INTEGER is
                -- Vertical position relative to parent
                -- (from EV_WIDGET)
        require -- from EV_WIDGET
                exists: notdestroyed;
                unmanaged: notmanaged
        do
                Result := implementation.y
        end;

feature -- Comparison

    same (other: like Current): BOOLEAN is
                -- Does Current widget and other correspond
                -- to the same screen object?
                -- (from EV_WIDGET)
        require -- from EV_WIDGET
                other_exists: other /= void
        do
                Result := other.implementation = implementation
        end;

feature -- Status report

    automatic_position: BOOLEAN;
                -- Does the widget take a new position when
                -- the parent resize ? (If it does, its size
                -- doesn’t changed). False by default
                -- (from EV_WIDGET)

    automatic_resize: BOOLEAN;
                -- Is the widget resized automatically when
                -- the parent resize ? In this case,
                -- automatic_position has no effect. True by
                -- default
                -- (from EV_WIDGET)

    destroyed: BOOLEAN is
                -- Is Current widget destroyed?
                -- (= implementation does not exist)
                -- (from EV_WIDGET)
        do
                Result := (implementation = void)
        end;

    insensitive: BOOLEAN is
                -- Is current widget insensitive to
                -- user actions? (If it is, events will
                -- not be dispatched to Current widget or
                -- any of its children)
                -- (from EV_WIDGET)
        require -- from EV_WIDGET
                exists: notdestroyed
        do
                Result := implementation.insensitive
        end;

    managed: BOOLEAN;
                -- Is the geometry of current widget managed by its
                -- container? This is the case always unless
                -- parent.manager = False (Always true except
                -- when the container is EV_FIXED). This is
                -- set in the procedure set_default
                -- (from EV_WIDGET)
    pressed: BOOLEAN is
            -- Is toggle pressed
            -- (from EV_TOGGLE_BUTTON)
        require -- from EV_TOGGLE_BUTTON
            exists: notdestroyed
        do
            Result := implementation.pressed
        end;

    shown: BOOLEAN is
            -- Is current widget visible?
            -- (from EV_WIDGET)
        require -- from EV_WIDGET
            exists: notdestroyed
        do
            Result := implementation.shown
        end;

feature -- Status setting

    destroy is
            -- Destroy actual screen object of Current
            -- widget and of all children.
            -- (from EV_WIDGET)
        do
            if notdestroyed then
                implementation.destroy;
                remove_implementation
            end
        ensure -- from EV_WIDGET
            destroyed: destroyed
        end;

    hide is
            -- Make widget and all children (recursively)
            -- invisible on the screen.
            -- (from EV_WIDGET)
        require -- from EV_WIDGET
            exists: notdestroyed
        do
            implementation.hide
        ensure -- from EV_WIDGET
            not_shown: notshown
        end;

    set_automatic_position (position: BOOLEAN) is
            -- Set automatic_position at position.
            -- (from EV_WIDGET)
    require -- from EV_WIDGET
        exists: notdestroyed
    do
        automatic_position := position
    ensure -- from EV_WIDGET
        automatic_position_set: automatic_position = position
    end;

set_automatic_resize (resize: BOOLEAN) is
        -- Set automatic_resize to resize.
        -- (from EV_WIDGET)
    require -- from EV_WIDGET
        exists: notdestroyed
    do
        automatic_resize := resize
    ensure -- from EV_WIDGET
        automatic_resize_set: automatic_resize = resize
    end;

set_center_alignment is
        -- Set text alignment of current label to center.
        -- (from EV_TEXT_CONTAINER)
    require -- from EV_TEXT_CONTAINER
        exists: notdestroyed
    do
        implementation.set_center_alignment
    end;

set_insensitive (flag: BOOLEAN) is
        -- Set current widget in insensitive mode if
        -- flag. This means that any events with an
        -- event type of KeyPress, KeyRelease,
        -- ButtonPress, ButtonRelease, MotionNotify,
        -- EnterNotify, LeaveNotify, FocusIn or
        -- FocusOut will not be dispatched to current
        -- widget and to all its children. Set it to
        -- sensitive mode otherwise.
        -- (from EV_WIDGET)
    require -- from EV_WIDGET
        exists: notdestroyed
    do
        implementation.set_insensitive (flag)
    ensure -- from EV_WIDGET
        flag = insensitive
    end;

set_left_alignment is
        -- Set text alignment of current label to left.
        -- (from EV_TEXT_CONTAINER)
        require -- from EV_TEXT_CONTAINER
                exists: notdestroyed
        do
                implementation.set_left_alignment
        end;

    set_pressed (button_pressed: BOOLEAN) is
                -- Set Current toggle on and set
                -- pressed to True.
                -- (from EV_TOGGLE_BUTTON)
        require -- from EV_TOGGLE_BUTTON
                exists: notdestroyed
        do
                implementation.set_pressed (button_pressed)
        ensure -- from EV_TOGGLE_BUTTON
                correct_state: pressed = button_pressed
        end;

    set_right_alignment is
                -- Set text alignment of current label to right.
                -- (from EV_TEXT_CONTAINER)
        require -- from EV_TEXT_CONTAINER
                exists: notdestroyed
        do
                implementation.set_right_alignment
        end;

    show is
                -- Make widget and all children (recursively)
                -- visible on the screen. (default)
                -- (from EV_WIDGET)
        require -- from EV_WIDGET
                exists: notdestroyed
        do
                implementation.show
        ensure -- from EV_WIDGET
                shown: shown
        end;

    toggle is
                -- Change the state of the toggel button to
                -- opposite
                -- (from EV_TOGGLE_BUTTON)
        require -- from EV_TOGGLE_BUTTON
                exists: notdestroyed
        do
                implementation.toggle
        ensure -- from EV_TOGGLE_BUTTON
                state_is_true: pressed = notold pressed
```

```
            end;

    feature -- Element change

        set_font (a_font: EV_FONT) is
                    -- Set font label to font_name.
                    -- (from EV_FONTABLE)
            require -- from EV_FONTABLE
                    exists: notdestroyed;
                    a_font_exists: a_font /= void;
                    a_font_specified: a_font.is_specified
            do
                    implementation.set_font (a_font)
            end;

        set_font_name (a_font_name: STRING) is
                    -- Set font label to a_font_name.
                    -- (from EV_FONTABLE)
            require -- from EV_FONTABLE
                    exists: notdestroyed;
                    a_font_name_exists: a_font_name /= void
            local
                    a_font: EV_FONT
            do
                    !! a_font.make;
                    a_font.set_name (a_font_name);
                    set_font (a_font)
            end;

        set_text (txt: STRING) is
                    -- Set text of current label to txt.
                    -- (from EV_TEXT_CONTAINER)
            require -- from EV_TEXT_CONTAINER
                    exists: notdestroyed;
                    not_a_text_void: txt /= void
            do
                    implementation.set_text (txt)
            ensure -- from EV_TEXT_CONTAINER
                    text_set: text.is_equal (txt)
            end;

    feature -- Resizing

        set_height (new_height: INTEGER) is
                    -- Set height to new_height.
                    -- (from EV_WIDGET)
            require -- from EV_WIDGET
                    exists: notdestroyed;
                    positive_height: new_height >= 0
        do
                implementation.set_height (new_height)
        ensure -- from EV_WIDGET
                dimensions_set: implementation.dimensions_set (width, new_height)
        end;

    set_maximum_height (max_height: INTEGER) is
                -- Set maximum_height to max_height.
                -- (from EV_WIDGET)
        require -- from EV_WIDGET
                exists: notdestroyed;
                large_enough: max_height >= 0
        do
                implementation.set_maximum_height (max_height)
        ensure -- from EV_WIDGET
                max_height = max_height
        end;

    set_maximum_width (max_width: INTEGER) is
                -- Set maximum_width to max_width.
                -- (from EV_WIDGET)
        require -- from EV_WIDGET
                exists: notdestroyed;
                large_enough: max_width >= 0
        do
                implementation.set_maximum_width (max_width)
        ensure -- from EV_WIDGET
                max_width = max_width
        end;

    set_minimum_height (min_height: INTEGER) is
                -- Set minimum__height to min_height.
                -- (from EV_WIDGET)
        require -- from EV_WIDGET
                exists: notdestroyed;
                large_enough: min_height >= 0
        do
                implementation.set_minimum_height (min_height)
        ensure -- from EV_WIDGET
                min_height = min_height
        end;

    set_minimum_size (min_width, min_height: INTEGER) is
                -- (from EV_WIDGET)
        require -- from EV_WIDGET
                exists: notdestroyed;
                large_enough: min_height >= 0;
                large_enough: min_width >= 0
        do
```

```
                    implementation.set_minimum_size (min_width, min_height)
            ensure -- from EV_WIDGET
                    min_width = min_width;
                    min_height = min_height
            end;

        set_minimum_width (min_width: INTEGER) is
                    -- Set minimum_width to min_width.
                    -- (from EV_WIDGET)
            require -- from EV_WIDGET
                    exists: notdestroyed;
                    large_enough: min_width >= 0
            do
                    implementation.set_minimum_width (min_width)
            ensure -- from EV_WIDGET
                    min_width = min_width
            end;

        set_size (new_width: INTEGER; new_height: INTEGER) is
                    -- Set width and height to new_width
                    -- and new_height.
                    -- (from EV_WIDGET)
            require -- from EV_WIDGET
                    exists: notdestroyed;
                    positive_width: new_width >= 0;
                    positive_height: new_height >= 0
            do
                    implementation.set_size (new_width, new_height)
            ensure -- from EV_WIDGET
                    dimensions_set: implementation.dimensions_set (new_width, new_
    height)
            end;

        set_width (new_width: INTEGER) is
                    -- Set width to new_width.
                    -- (from EV_WIDGET)
            require -- from EV_WIDGET
                    exists: notdestroyed;
                    positive_width: new_width >= 0
            do
                    implementation.set_width (new_width)
            ensure -- from EV_WIDGET
                    dimensions_set: implementation.dimensions_set (new_width, height)
            end;

        set_x (new_x: INTEGER) is
                    -- Set horizontal position relative
                    -- to parent to new_x.
                    -- (from EV_WIDGET)
        require -- from EV_WIDGET
            exists: notdestroyed;
            unmanaged: notmanaged
        do
            implementation.set_x (new_x)
        ensure -- from EV_WIDGET
            x_set: x = new_x
        end;

    set_x_y (new_x: INTEGER; new_y: INTEGER) is
            -- Set horizontal position and
            -- vertical position relative to parent
            -- to new_x and new_y.
            -- (from EV_WIDGET)
        require -- from EV_WIDGET
            exists: notdestroyed;
            unmanaged: notmanaged
        do
            implementation.set_x_y (new_x, new_y)
        end;

    set_y (new_y: INTEGER) is
            -- Set vertical position relative
            -- to parent to new_y.
            -- (from EV_WIDGET)
        require -- from EV_WIDGET
            exists: notdestroyed;
            unmanaged: notmanaged
        do
            implementation.set_y (new_y)
        ensure -- from EV_WIDGET
            y_set: y = new_y
        end;

feature {NONE} -- Implementation

    implementation: EV_TOGGLE_BUTTON_I;
            -- (from EV_TOGGLE_BUTTON)

    remove_implementation is
            -- Remove implementation of Current widget.
            -- (from EV_WIDGET)
        do
            implementation := void
        ensure -- from EV_WIDGET
            void_implementation: implementation = void
        end;

    set_default is
```

```
                        -- Do the necessary initialization after
                        -- creation
                        -- Set default values of Current widget.
                        -- (from EV_WIDGET)
                do
                        implementation.build
                        parent.add_child (Current)
                        managed := parent.manager
                end;

        set_font_imp (an_implementation: EV_FONTABLE_I) is
                        -- Set implementation to an_implementation.
                        -- (from EV_FONTABLE)
                require -- from EV_FONTABLE
                        an_implementation_exists: an_implementation /= void
                do
                        implementation := an_implementation
                end;

feature -- Event - command association

        add_button_press_command (mouse_button: INTEGER; command: EV_COM-
MAND; arguments: EV_ARGUMENTS) is
                        -- Add 'command' to the list of commands to
                        -- be executed when button no 'mouse_button'
                        -- is pressed
                        -- (from EV_WIDGET)
                do
                        implementation.add_button_press_command (mouse_button, com-
mand, arguments)
                end;

        add_button_release_command (mouse_button: INTEGER; command: EV_COM-
MAND; arguments: EV_ARGUMENTS) is
                        -- Add 'command' to the list of commands to
                        -- be executed when button no 'mouse_button'
                        -- is released
                        -- (from EV_WIDGET)
                do
                        implementation.add_button_release_command (mouse_button, com-
mand, arguments)
                end;

        add_click_command (command: EV_COMMAND; arguments: EV_ARGUMENTS) is
                        -- Add 'command' to the list of commands to be
                        -- executed when the button is pressed
                        -- (from EV_BUTTON)
                require -- from EV_BUTTON
                        valid_command: command /= void
                do
                        implementation.add_click_command (command, arguments)
                end;

        add_delete_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)
is
                        -- (from EV_WIDGET)
                do
                        implementation.add_delete_command (command, arguments)
                end;

        add_double_click_command (mouse_button: INTEGER; command: EV_COMMAND;
arguments: EV_ARGUMENTS) is
                        -- Add 'command' to the list of commands to
                        -- be executed when button no 'mouse_button'
                        -- is double clicked
                        -- (from EV_WIDGET)
                do
                        implementation.add_double_click_command (mouse_button, com-
mand, arguments)
                end;

        add_enter_notify_command (command: EV_COMMAND; arguments: EV_ARGU-
MENTS) is
                        -- (from EV_WIDGET)
                do
                        implementation.add_enter_notify_command (command, arguments)
                end;

        add_expose_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)
is
                        -- (from EV_WIDGET)
                do
                        implementation.add_expose_command (command, arguments)
                end;

        add_key_press_command (command: EV_COMMAND; arguments: EV_ARGU-
MENTS) is
                        -- (from EV_WIDGET)
                do
                        implementation.add_key_press_command (command, arguments)
                end;

        add_key_release_command (command: EV_COMMAND; arguments: EV_ARGU-
MENTS) is
                        -- (from EV_WIDGET)
                do
                        implementation.add_key_release_command (command, arguments)
                end;
        add_leave_notify_command (command: EV_COMMAND; arguments: EV_ARGU-
MENTS) is
                        -- (from EV_WIDGET)
                do
                        implementation.add_leave_notify_command (command, arguments)
                end;

        add_motion_notify_command (command: EV_COMMAND; arguments: EV_ARGU-
MENTS) is
                        -- (from EV_WIDGET)
                do
                        implementation.add_motion_notify_command (command, arguments)
                end;

        add_toggle_command (command: EV_COMMAND; arguments: EV_ARGUMENTS)
is
                        -- Add 'command' to the list of commands to be
                        -- executed when the button is toggled
                        -- (from EV_TOGGLE_BUTTON)
                require -- from EV_TOGGLE_BUTTON
                        valid_command: command /= void
                do
                        implementation.add_toggle_command (command, arguments)
                end;

        last_command_id: INTEGER is
                        -- Id of the last command added by feature
                        -- 'add_command'
                        -- (from EV_WIDGET)
                require -- from EV_WIDGET
                        exists: notdestroyed
                do
                        Result := implementation.last_command_id
                end;

        remove_command (command_id: INTEGER) is
                        -- Remove the command associated with
                        -- 'command_id' from the list of actions for
                        -- this context. If there is no command
                        -- associated with 'command_id', nothing
                        -- happens.
                        -- (from EV_WIDGET)
                require -- from EV_WIDGET
                        exists: notdestroyed
                do
                        implementation.remove_command (command_id)
                end;
```

```
invariant

                -- from GENERAL
        reflexive_equality: standard_is_equal (Current);
        reflexive_conformance: conforms_to (Current);

end -- class MAIN_WINDOW_BUTTON
```

Figure 3.12.1 shows a screenshot of the example.

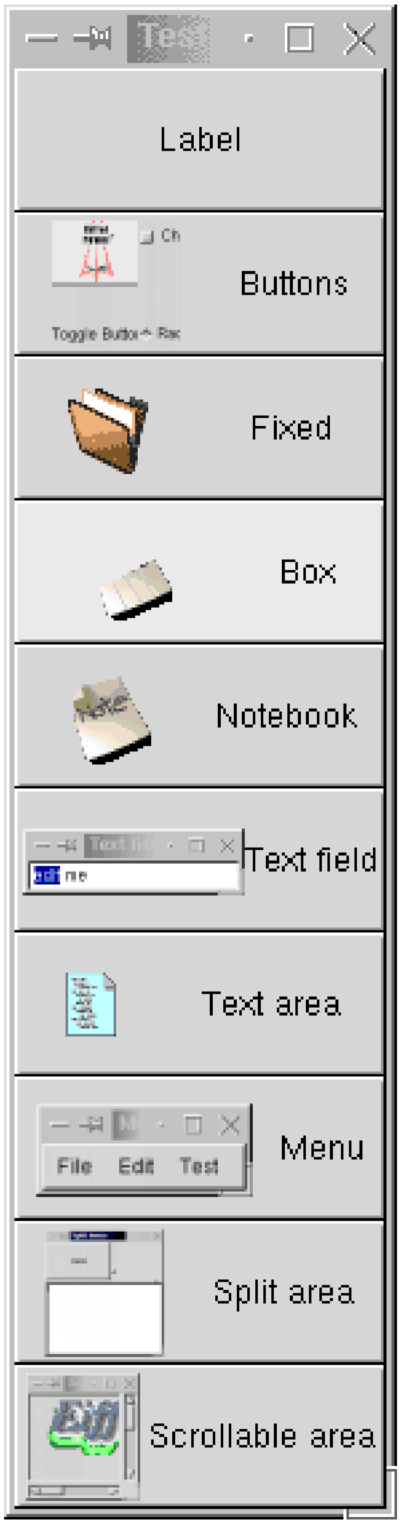


Figure 3.12.1 Main windoindexing