The new EiffelVision Library

version 4.3 (draft)

Interactive Software Engineering

Manual identification

The New EiffelVision Library, ISE Technical Report TR-EI-65/NE.

Publication history

First published in draft form: April 1999 (this version). Corresponds to release 4.3 of the ISE Eiffel environment.

Authors

Sami Kallio, Leila Ait Kaci, with help from Paulette LeBlanc.

Software credits

Leila Ait Kaci, Sami Kallio, Sam O'Connor.

Cover design

Rich Ayling.

Copyright notice and proprietary information

Copyright © Interactive Software Engineering Inc. (ISE), 1999. May not be reproduced in any form (including electronic storage) without the written permission of ISE. "Eiffel Power" and the Eiffel Power logo are trademarks of ISE.

All uses of the product documented here are subject to the terms and conditions of the ISE Eiffel user license. Any other use or duplication is a violation of the applicable laws on copyright, trade secrets and intellectual property.

Special duplication permission for educational institutions

Degree-granting educational institutions using ISE Eiffel for teaching purposes as part of the Eiffel University Partnership Program may be permitted under certain conditions to copy specific parts of this book. Contact ISE for details.

About ISE

ISE (Interactive Software Engineering) is dedicated to improving software quality and productivity through advanced methods, tools and languages, based on sound scientific principles and on the systematic application of object technology.

The company provides a complete line of development tools as well as on-site consulting, library development services, and a training program on all aspects of O-O technology: analysis, design, implementation techniques, graphics, library building, Eiffel language, project management, large system design etc.

ISE is the original designer of the Eiffel method and language and a member of NICE, the Nonprofit International Consortium for Eiffel.

For more information

Interactive Software Engineering Inc.
ISE Building, 2nd floor
270 Storke Road
Goleta, CA 93117 USA
Telephone 805-685-1006, Fax 805-685-6869

Internet and e-mail

ISE maintains a rich source of information at **http://eiffel.com**, with more than 1200 Web pages including online documentation, downloadable files, product descriptions, links to ISE partners, University Partnership program, mailing list archives, announcements, press coverage, Frequently Asked Questions, Support pages, and much more.

Write to **info@eiffel.com** for information about products and services. Write to **userlist-request@eiffel.com** to subscribe to the ISE Eiffel user list.

Support programs

ISE offers a variety of support options tailored to the diverse needs of its customers. Write to **info@eiffel.com** or check the support pages at **http://eiffel.com** for details.

Preface: Why EiffelVision?

The EiffelVision library offers an object-oriented framework for graphical user interface (GUI) development. Using EiffelVision, developers can access all necessary GUI components — “widgets”, “dialogs” and “controls”as well as truly graphical elements such as figures, points, lines, arcs, polygons and the like — to develop a modern, functional and good-looking graphical interactive application.

EiffelVision has played a major role in ISE Eiffel and provided numerous Eiffel projects with a powerful, portable graphics development platform. To reflect the advances in graphical development toolkits, a “New EiffelVision” library development was undertaken in 1998 and the first version made available with ISE Eiffel 4.3 in March of 1999. This document serves as functional specification and user reference for the new EiffelVision library.

EiffelVision scope

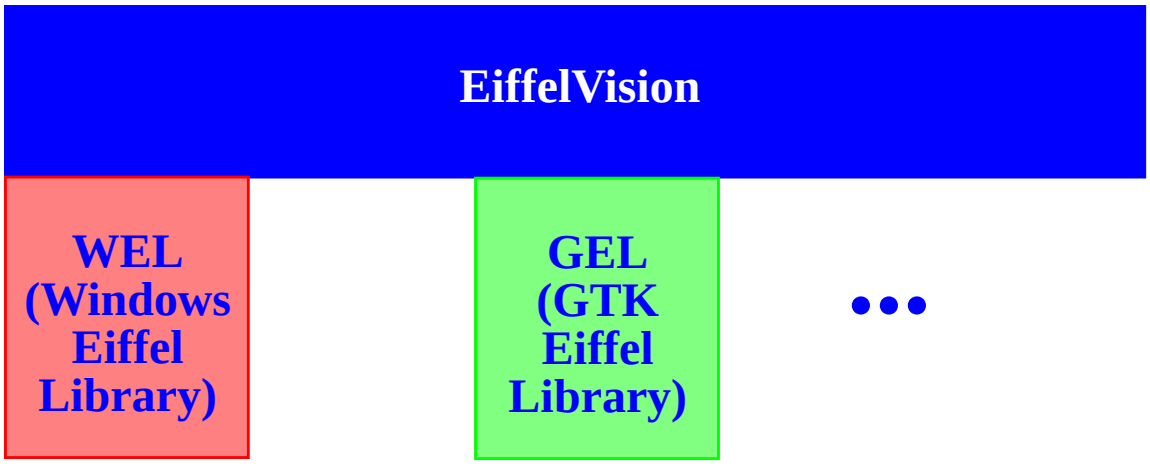
The EiffelVision library addresses all the major needs of developers of systems supporting modern graphical interfaces. EiffelVision runs on Microsoft Windows NT, Windows 95/98, all major Unix platforms (including Linux) and VMS. All versions are fully source-compatible; with only a recompile, applications will run on every supported platform with the native look-and-feel.

EiffelVision provides an effective way of building advanced graphical applications using user interface standards and toolkits (such as Microsoft Windows and GTK) without having to learn the details of the toolkits, their API and their C interfaces. Instead, you can use EiffelVision to work entirely in terms of high level abstractions representing windows, resources, graphical figures, menus, buttons etc., and apply clearly understandable operations to the corresponding objects.

EiffelVision supports both interface objects and graphical figures such as circles and polygons, as well as composite figures.

EiffelVision architecture

EiffelVision relies on a two-tiered architecture illustrated by the following figure.



The two tiers play complementary roles:

- At the top level, EiffelVision provides fully portable graphics.
- At the lower level, platform-specific libraries cover the graphical mechanisms of graphics platforms such as Windows and X.

The lower tier serves for the implementation of the upper tier, but can also be used independently. For example WEL has had a resounding success with Windows developers who need an advanced mechanism for building Windows-specific graphical applications, taking advantage of every facility of the Windows API (Application Programming Interface) and of the Eiffel approach, but do not need portability on the client side.

This flexibility of the EiffelVision architecture, enabling users to work at their level of choice — EiffelVision level for portable graphics, lower tier for platform-specific development, and possibly a mix of the two — has proved to be a key attraction of the library.

The new EiffelVision and its benefits

EiffelVision was originally built as an Eiffel wrapper around an early version of the Motif toolkit for X Windows. The introduction of a Windows version led to the two-tier architecture shown above, and to the development of the WEL library, which has proved on its own to be one of the most attractive components of the ISE Eiffel offering. The Motif part was covered by MEL (Motif Eiffel Library).

In 1998 ISE made the decision to update the design of EiffelVision to reflect advances in GUI technology on various platforms. The new EiffelVision retains the principles described above and follows the EiffelVision of powerful portable graphics; it develops and enhances these original ideas through a number of major advances.

Thanks to the new EiffelVision developers of GUI applications can benefit from a number of improvements:

- The new EiffelVision gives developers full access to the **powerful widgets** (controls) available in modern graphical toolkits, such as tree views, multi-column lists (list views), notebooks, status bars, advanced dialogs and many others from both Windows and X toolkits.
- The library relies on a much simplified model of the relationships between the widgets, removing the notion of "attachment" and introducing a simple notion of "**container**" for defining the relative placement of widgets within a window.
- The new EiffelVision **optimizes memory usage** by avoiding the duplication of information. This enables developers to build "lean and mean" graphical applications.
- The command model, managing the relationship between an application's user interface and its semantic model, has been greatly improved and simplified thanks to the use of Eiffel's new **agent** mechanism.
- On the X side, EiffelVision now relies on the **GTK** toolkit, a high-quality library providing numerous widgets and many other useful facilities.
- Supporting the new EiffelVision, ISE is developing a new version of the **EiffelBuild** interactive application builder (not described further in this manual), providing powerful facilities for building the EiffelVision-based GUI component of an application directly from the application's processing classes.

Scope

Throughout this document, the terms "EiffelVision" and "the library" refer to the new library, while 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.

Definitions, acronyms and abbreviations

Eiffel — object-oriented language and method, based on Design by Contract principles and described in the book *Eiffel : The Language* [Meyer 1992].

EiffelVision — object oriented GUI and graphic library for application development. In this manual, denotes the new version of EiffelVision introduced with ISE Eiffel release 4.3.

Old EiffelVision — Older version of Eiffel Vision (still available under 4.3).

GTK— The General Image Manipulation Program (GIMP) tookit. See [GTK 1998].

Widget — GUI component in EiffelVision. Also called "control" or "context".

References

[Meyer 1992]

Bertrand Meyer; *Eiffel: The Language*; Prentice Hall Object-Oriented Series, 1991; second revised printing, 1992. See **http://eiffel.com/doc/documentation.html#etl**.

[Meyer 1997]

Bertrand Meyer; *Object-Oriented Software Construction, second edition*; Prentice Hall Object-Oriented Series, 1997. See **http://eiffel.com/doc/oosc.html**.

[GTK 1998]

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

Status of this document and of the library

This manual is a draft and you will notice that some sections are sketched or missing. The current version of the new EiffelVision library is still a beta version. Not all the functionalities described in this manual are implemented; conversely, not all implemented functionalities are documented.

We will be grateful for any problem report. Please use the EiffelVision Talkitover group at **http://talkitover.com/vision** to report problems and discuss future evolutions of EiffelVision.

Prerequisites

The manual assumes that you have a reasonably good knowledge of Eiffel and object-oriented software development and a good understanding of the basic conceps of Graphical User Interfaces. Platform-specific knowledge of GUI programming is not necessary.

Organisation of the manual

Chapter 1, “The EiffelVision model”, describes the general mechanisms and principles used by the library.

Chapter 2, “EiffelVision basics”, contains the detailed description of the EiffelVision kernel.

Chapter 3, “Containers”, Chapter 4, “Primitives”, Chapter 5, “Items”, Chapter 6, “Components” and Chapter 7, “Standard Dialogs” contain the detailed description of the containers, primitives, items, components and standard dialogs.

Contents

1

The EiffelVision model

EiffelVision is an abstract, multi-platform library that provides components for building the GUI for an application and drawing figures onto the screen.

Supported platforms include :

- Windows NT / 95 / 98.
- Unix, Linux and all other platforms supported by the GTK toolkit.

EiffelVision contains a set of GUI components and methods to associate actions with GUI events..

Most of the widgets in EiffelVision have events to which you can associate commands. For example, there is an action **Button_press** for a button widget.

1.1 Events

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

Simple examples of events are mouse button clickes and keystrokes. Others include timer activation, mouse movement, auto-repeating keyboard, context resizing and changing 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. As a result part of which defines an application is the two-dimensional grid of what events are meaningful in what context — a **state domain**. The following diagram is a simple example of state domain.

Event / **Context**	Left click	Right click	Cursor out	Keyboard
Window_1			•	•
Window_2				•
Button_1	•	•	•	

The • mark denotes the entries for which the given event is meaningful. For example, the Cursor out event (which occurs when you move the cursor out of itscurrent context) is meaningful for *Window_1* and *Button_1* but not for *Window_2*.

As execution progresses, the state domain can change. Because a typical application can give you the choice between several possible events in several contexts, the state domain can be quite large. However a certain operation can trigger a confirmation panel in which the *application* only recognizes two events: clicking either the **OK** or **Cancel** button. In this case, the application enteres a new, smaller state domain.

1.2 Commands

When an event occurs in a certain context and the context-event pair is part of the current state domain, the application executes a certain action. That action is represented in Eiffel by an *object* — an instance of the EiffelVision class *EV_COMMAND*. More precisely, it is 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 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 you. This avoids 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 (where every command will be an object) and provides added power. In addition to the *execute* procedure, which describes the execution of the command and corresponds to the callback, command objects can 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* [Meyer 1997].

The following 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 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:

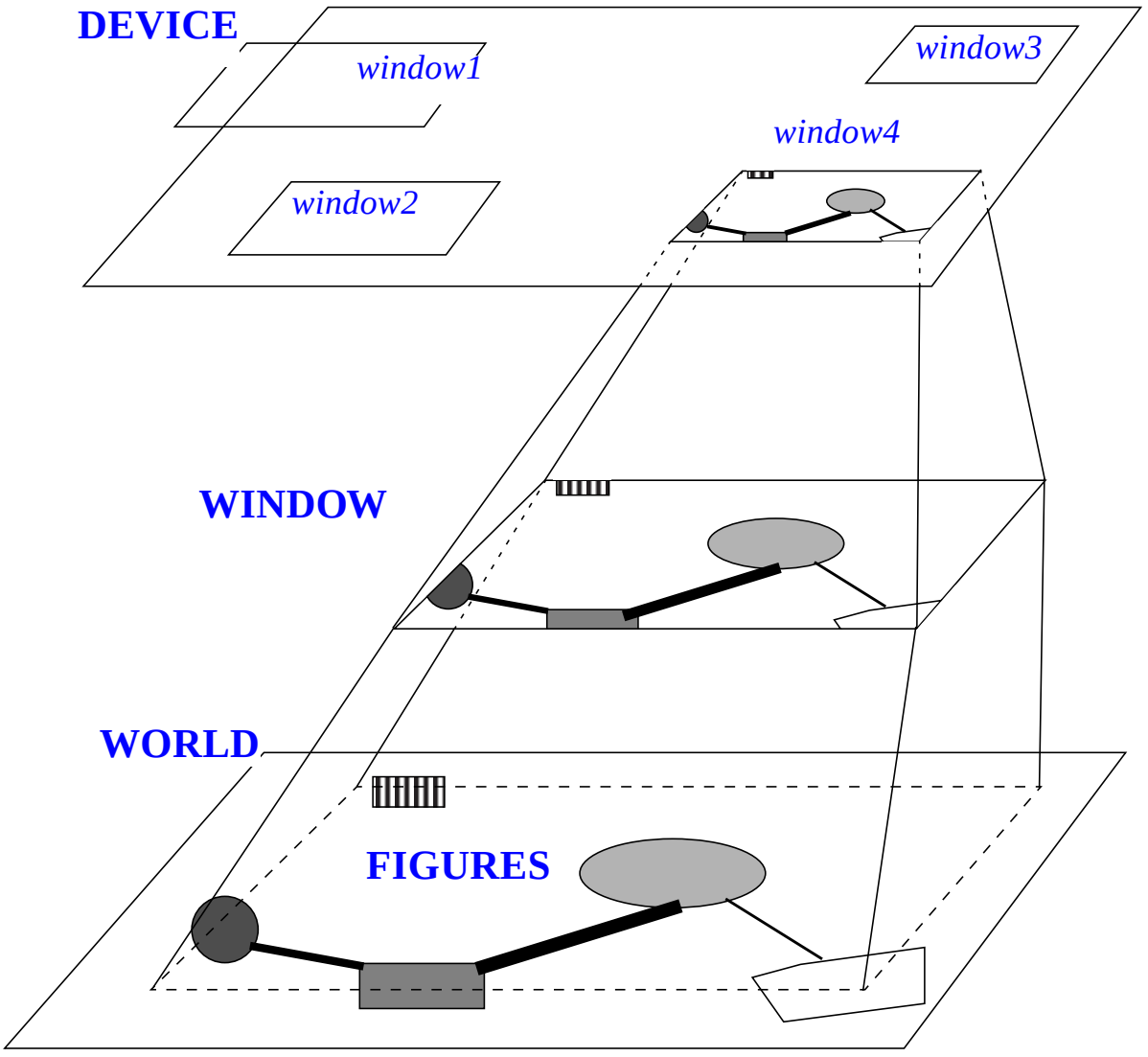
```
history_list.item.undo;
history_list.back
```

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

1.3 Figures

The interface of an EiffelVision application can also include graphical figures.

The EiffelVision model figures are based on a familiar notion: geographical maps. The design of a map uses several levels of abstraction, illustrated on the following figure.



The Graphical Model

You can view the reality behind the model (in an already abstracted form) as a set of geometrical shapes or **figures**. On a map the figures represent rivers, roads, towns and other geographical objects. Using this schema, 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 a country, and subsidiary windows devoted to large cities or outlying parts (as with Hawaii in maps of the USA).
- 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.

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 display on rectangular areas of the device (the computer screen).

The figure later in this section 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. The windows can 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:

- You can 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).
- You can 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).

1.4 Drag and Drop

To be completed

1.5 Pick and Drop

To be completed

2

EiffelVision basics

This chapter describes the kernel of the EiffelVision library and the basic mechanism to use its classes.

2.1 Widgets

As mentioned in the previous chapter, a widget is a basic graphical object. In Eiffel Vision, there are two kinds of widgets :

- Containers — widget that allows other widgets to be put inside itself.
- Primitives — widget that cannot accept any widget inside itself.

Creation

Creating a widget is simple and consistent. All the widgets except the multi-column list have the creation procedure *make*:

make (par: EV_CONTAINER)

Procedure *make* has one argument, *par*, an EiffelVision container denoting the widget's parent. If *par* is void, the widget does not appear graphically at creation, but only when you give it a parent later on during execution using the *set_parent* procedure. A widget without a parent will not be destroyed; to avoid memory leaks, you should destroy all unnecessary widgets without parents.

Procedure *make* creates the widget using default setting for the specific type of widget. Some widgets have additional creation procedures, which you can use when you need finer control over widget creation.

Management

There are two different types of behavior for a widget:

- Managed
- Unmanaged

This behavior depends on the parent's type.

An attribute *managed* determines which behavior type the widget follows. An unmanaged widget freely chooses its size, minimum size and position — the parent has no effect on these parameters. On the contrary, a managed widget can only set its minimum size — the size and position of the widget depends directly of those of the parent.

Most of the containers manage the behavior of their children. The only exceptions are the **EV_FIXED** and **EV_SCROLLABLE_AREA**.

However, a managed widget can choose its behavior inside the parent.

A manager container actually manages cells. Each child widget is in one cell. These cells automatically take the size and position the container gives them, except if the widget inside has the option *expandable* set to *False*. In this case, neither the cell or widget resizes and the container reserves the resting space for the other cells.

Inside these cells, the widgets have a certain freedom set by two options : *horizontal_resizable* and *vertical_resizable*. These options determine if the widget will resize itself inside its cell or not. If the widget does not change size, then the widget moves into the center of the cell.

class

```
indexing
        description: "EiffelVision widget. Most general notion of widget (i.e. user interface object)."
        status: "See notice at end of class"
        names: widget
        date: "$Date: 1999/03/19 20:19:00 $"
        revision: "$Revision: 1.36 $"

deferred class interface
        EV_WIDGET

feature -- Access

        parent: EV_WIDGET
                -- The parent of the Current widget
                -- Can be void.
            require
                exists: not destroyed

feature -- Measurement

        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

    y: INTEGER
            -- Vertical position relative to parent
        require
            exists: not destroyed

feature -- Status report

    background_color: EV_COLOR
            -- Color used for the background of the widget
        require
```

```
            exists: not destroyed
        ensure
            valid_result: Result /= void

    expandable: BOOLEAN
            -- Does the widget expand its cell to take the
            -- size the parent would like to give to it.
        require
            exists: not destroyed

    foreground_color: EV_COLOR
            -- Color used for the foreground of the widget
            -- usually the text.
        require
            exists: not destroyed
        ensure
            valid_result: Result /= void

    horizontal_resizable: BOOLEAN
            -- Does the widget change its width when the parent
            -- or the user want to resize the widget
        require
            exists: not destroyed

    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: **not** destroyed

vertical_resizable: BOOLEAN
-- Does the widget change its width when the parent
-- or the user want to resize the widget
require
exists: **not** destroyed

feature -- Status setting

hide
-- Make widget invisible on the screen.
require
exists: **not** destroyed
ensure
not_shown: **not** shown

set_default_colors
-- Initialize the colors of the widget
require
exists: **not** destroyed

set_default_minimum_size
-- Initialize the size of the widget.
-- Redefine by some widgets.
require
exists: **not** destroyed

set_default_options
-- Initialize the options of the widget.
require
exists: **not** destroyed

set_expand (flag: BOOLEAN)
-- Make flag the new expand option.
require
exists: **not** destroyed

set_focus
-- Set focus to Current
require
exists: **not** destroyed

```
        set_horizontal_resize (flag: BOOLEAN)
                -- Adapt resize_type to flag.
            require
                exists: not destroyed
            ensure
                horizontal_resize_set: horizontal_resizable = flag

        set_insensitive (flag: BOOLEAN)
                -- Set current widget in insensitive mode.
                -- 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.
            require
                exists: not destroyed
            ensure
                insensitive = flag

        set_vertical_resize (flag: BOOLEAN)
                -- Adapt resize_type to flag.
            require
                exists: not destroyed
            ensure
                vertical_resize_set: vertical_resizable = flag

        show
                -- Make widget visible on the screen. (default)
                -- Do nothing if the widget has no parent.
            require
                exists: not destroyed;
                has_parent: parent /= void
            ensure
                shown: (parent /= void) implies shown

feature -- Element change

        set_background_color (color: EV_COLOR)
                -- Make color the new background_color
            require
```

```
            exists: not destroyed;
            valid_color: is_valid (color)
        ensure
            background_color_set: background_color = color

    set_foreground_color (color: EV_COLOR)
            -- Make color the new foreground_color
        require
            exists: not destroyed;
            valid_color: is_valid (color)
        ensure
            foreground_color_set: foreground_color = color

    set_parent (par: EV_CONTAINER)
            -- Make par the new parent of the widget.
            -- par can be Void then the parent is the screen.
        require
            exists: not destroyed
        ensure
            parent_set: parent = par

feature -- Resizing

    set_height (value: INTEGER)
            -- Make value the new height.
            -- widget must be unmanaged.
        require
            exists: not destroyed;
            unmanaged: not managed;
            positive_height: value >= 0
        ensure
            dimensions_set: implementation.dimensions_set (width, value)

    set_minimum_height (value: INTEGER)
            -- Make value the new minimum_height.
        require
            exists: not destroyed;
            large_enough: value >= 0
        ensure
            minimum_height_set: implementation.minimum_height_set (value)

    set_minimum_size (min_width, min_height: INTEGER)
```

```
            -- Make min_width the new minimum_width
            -- and min_height the new minimum_height.
        require
            exists: not destroyed;
            large_enough: min_height >= 0;
            large_enough: min_width >= 0
        ensure
            minimum_dimension_set: implementation.minimum_dimensions_set (min_width,
min_height)

    set_minimum_width (value: INTEGER)
            -- Make value the new minimum_width.
        require
            exists: not destroyed;
            large_enough: value >= 0
        ensure
            minimum_width_set: implementation.minimum_width_set (value)

    set_size (new_width: INTEGER; new_height: INTEGER)
            -- Make new_width the new width
            -- and new_height the new height.
            -- widget must be unmanaged.
        require
            exists: not destroyed;
            unmanaged: not managed;
            positive_width: new_width >= 0;
            positive_height: new_height >= 0
        ensure
            dimensions_set: implementation.dimensions_set (new_width, new_height)

    set_width (value: INTEGER)
            -- Make value the new width.
            -- widget must be unmanaged.
        require
            exists: not destroyed;
            unmanaged: not managed;
            positive_width: value >= 0
        ensure
            dimensions_set: implementation.dimensions_set (value, height)

    set_x (value: INTEGER)
            -- Put at horizontal position value relative
```

```
            -- to parent.
        require
            exists: not destroyed;
            unmanaged: not managed
        ensure
            x_set: implementation.x_set (value)

    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
        ensure
            x_y_set: implementation.position_set (new_x, new_y)

    set_y (value: INTEGER)
            -- Put at vertical position value relative
            -- to parent.
        require
            exists: not destroyed;
            unmanaged: not managed
        ensure
            y_set: implementation.y_set (x)

feature -- Event - command association

    add_button_press_command (mouse_button: INTEGER; cmd: EV_COMMAND; arg:
EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed
            -- when button number 'mouse_button' is pressed.
        require
            exists: not destroyed;
            valid_command: cmd /= void

    add_button_release_command (mouse_button: INTEGER; cmd: EV_COMMAND; arg:
EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed
            -- when button number 'mouse_button' is released.
        require
            exists: not destroyed;
            valid_command: cmd /= void
```

```
add_destroy_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed
        -- when the widget is destroyed.
    require
        exists: not destroyed;
        valid_command: cmd /= void

add_double_click_command (mouse_button: INTEGER; cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed
        -- when button number mouse_button is double
        -- clicked.
    require
        exists: not destroyed;
        valid_command: cmd /= void

add_enter_notify_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed
        -- when the cursor of the mouse enter the widget.
    require
        exists: not destroyed;
        valid_command: cmd /= void

add_expose_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd 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;
        valid_command: cmd /= void

add_get_focus_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed
        -- when the widget get the focus.
    require
        exists: not destroyed;
        valid_command: cmd /= void

add_key_press_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed when
        -- Add cmd 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;
            valid_command: cmd /= void

    add_key_release_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add cmd 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;
            valid_command: cmd /= void

    add_leave_notify_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed
            -- when the cursor of the mouse leave the widget.
        require
            exists: not destroyed;
            valid_command: cmd /= void

    add_loose_focus_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed
            -- when the widget loose the focus.
        require
            exists: not destroyed;
            valid_command: cmd /= void

    add_motion_notify_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed
            -- when mouse move.
        require
            exists: not destroyed;
            valid_command: cmd /= void

feature -- Event -- removing command association

    remove_button_press_commands (mouse_button: INTEGER)
            -- Empty the list of commands to be executed when
            -- button number 'mouse_button' is pressed.
        require
            exists: not destroyed
```

```
remove_button_release_commands (mouse_button: INTEGER)
        -- Empty the list of commands to be executed when
        -- button number 'mouse_button' is released.
    require
        exists: not destroyed

remove_destroy_commands
        -- Empty the list of commands to be executed when
        -- the widget is destroyed.
    require
        exists: not destroyed

remove_double_click_commands (mouse_button: INTEGER)
        -- Empty the list of commands to be executed when
        -- button number 'mouse_button' is double clicked.
    require
        exists: not destroyed

remove_enter_notify_commands
        -- Empty the list of commands to be executed when
        -- the cursor of the mouse enter the widget.
    require
        exists: not destroyed

remove_expose_commands
        -- Empty 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

remove_get_focus_commands
        -- Empty the list of commands to be executed when
        -- the widget get the focus.
    require
        exists: not destroyed

remove_key_press_commands
        -- Empty 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

    remove_key_release_commands
            -- Empty 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

    remove_leave_notify_commands
            -- Empty the list of commands to be executed when
            -- the cursor of the mouse leave the widget.
        require
            exists: not destroyed

    remove_loose_focus_commands
            -- Empty the list of commands to be executed when
            -- the widget loose the focus.
        require
            exists: not destroyed

    remove_motion_notify_commands
            -- Empty the list of commands to be executed when
            -- the mouse move.
        require
            exists: not destroyed

end -- class EV_WIDGET
```

2.2 Events

There are two different types of events in EiffelVision :

- general events — common to all widgets.
- widget-specific events.

General Events

The following list describes the general events.

button press — a mouse button has been pressed over the widget.

button released — a mouse button has been released over the widget.

double click — a mouse button has been double clicked over the widget.

motion notify — mouse pointer has been moved over the widget.

delete — the widget has been deleted.

expose — a part of the widget has been redrawn because it was exposed.

key press — a key has been pressed over the widget.

key release — a key has been released over the widget.

enter notify — the mouse pointer has entered the area of the widget.

leave notify — the mouse pointer has left the area of the widget.

Widget-specific events

As the name suggests, these events are specific to a given widget. For example, a button click event thet occurs when the button widget is clicked.

2.3 Event Data

Event data is an object given as an argument of the *execute* procedure of a command. This information is specific to an event: for example, the location of the mouse pointer. Widget specific events do not contain any event data. You will find different types of event data:

EV_EVENT_DATA

indexing

description: "EiffelVision event data. Information given byEiffelVision when a callback 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 1.9 1999/03/12 20:10:54 aitkaci Exp $"
date: "$Date: 1999/03/12 20:10:54 $"
revision: "$Revision: 1.9 $"

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*

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 1.7 1998/10/19 17:26:43 aitkaci Exp $"
date: "$Date: 1998/10/19 17:26:43 $"
revision: "$Revision: 1.7 $"

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*

EV_MOTION_EVENT_DATA

This class represents event data for the mouse motion event.

```
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 1.6 1998/10/19 17:26:45 aitkaci Exp $"
    date: "$Date: 1998/10/19 17:26:45 $"
    revision: "$Revision: 1.6 $"

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
```

EV_KEY_EVENT_DATA

This class represents event data for the keyboard events : key press or key release.

```
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 1.2 1998/10/19 17:26:44 aitkaci Exp $"
date: "$Date: 1998/10/19 17:26:44 $"
revision: "$Revision: 1.2 $"

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*

2.4 Commands

You can add as many commands as necessary to an event — the commands execute in the same order than they were added. The feature *remove_?_commands* removes all commands attached to the given event. A feature *remove_command* will be available to remove only one given command.

Class

indexing
description: "General notion of command (semantic unity).To write an actual command inherit from thisclass and implement the 'execute%' feature"
status: "See notice at end of class"
date: "$Date: 1999/03/04 19:12:02 $"
revision: "$Revision: 1.6 $"

deferred class interface
EV_COMMAND

feature *-- Basic operations*

execute (args: EV_ARGUMENT; data: EV_EVENT_DATA)
-- Execute Current command.
-- args and data are automatically passed by
-- EiffelVision when Current command is
-- invoked as a callback.

end *-- class EV_COMMAND*

2.5 Undoable Command

An undoable command is a command that include an undoable mechanism.

Class

2.6 Routine Command

A routine command is a command created through an agent. This type of routines can have several *execute* functions in the same class. However, the procedure that creates the command must have the same signature that the execute function of a command :

execute (arg: EV_ARGUMENT; data: EV_EVENT_DATA)

Class

indexing
" Routine notion of command. To create this kind of command any procedure with the following signature : execute (arg: EV_ARGUMENT; event_data: EV_EVENT_DATA) can be used."
status: "See notice at end of class"
date: "$Date: 1999/03/02 17:13:08 $"
revision: "$Revision: 1.1 $"

class interface
EV_ROUTINE_COMMAND

creation
make

feature *-- Access*

procedure: PROCEDURE [ANY, TUPLE [EV_ARGUMENT, EV_EVENT_DATA]]

feature *-- Basic operations*

execute (args: EV_ARGUMENT; data: EV_EVENT_DATA)
-- Execute Current command.
-- args and data are automatically passed by
-- EiffelVision when Current command is
-- invoked as a callback.
-- Call the routine

end *-- class EV_ROUTINE_COMMAND*

2.7 Arguments

The *execute* procedure of a command class ask for an **EV_ARGUMENT** in its arguments. There are 4 kind of arguments implemented in EiffelVision, but any class that inherits from **EV_ARGUMENT** becomes an argument :

- One, two or three parameter arguments.
- Tuple arguments.

One, two or three parameters

For an argument with only one, two or three parameters, you can use the **EV_ARGUMENT1**, **EV_ARGUMENT2** and **EV_ARGUMENT3** types. You create this type of argument by giving the data to store to the creation routine *make*. You can then retrieve the data using the features *first*, *second* and *third*.

indexing
description: "EiffelVision EV_ARGUMENT2. To be used when passing two arguments to a command."
status: "See notice at end of class"
id: "$Id: ev_argument2.e,v 1.3 1999/03/02 17:50:38 aitkaci Exp $"
date: "$Date: 1999/03/02 17:50:38 $"
revision: "$Revision: 1.3 $"

class interface
EV_ARGUMENT2 [G, H]

creation
make

feature *-- Initialization*

```
        make (first_element: G; second_element: H)
                -- Create an argument with first_element and
                -- second_element.

feature -- Access

        second: H
                -- Second element of the argument

end -- class EV_ARGUMENT2
```

Tuple arguments

The tuple argument type allows you to give more data. You can create an **EV_TUPLE_ARGUMENT** by giving a tuple to the argument.

2.8 Timers

To be completed

2.9 Colors

X Windows System provides color map handling, so that the closest color is return.

2.10 Fonts

To be completed

3

Containers

A *container* is a widget that allows other widgets, called its *children* to be put inside the container. 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.

Usually container manages its children. This means that the size and position of a child is specified by the container. The child can only specify its size and location under the restrictions of the container. In a non-manager container, a widget freely sets its position, size and minimum size, while in a manager container, a widget can only set its minimum size. In this case, the container gives the position and size of the widge. The attributes *automatic_position* and *automatic_resize* of **EV_WIDGET** control the behavior of the child inside the container.

Class

```
indexing
        description: "EiffelVision container. Container is a widget that can hold children inside it"
        status: "See notice at end of class"
        id: "$Id: ev_container.e,v 1.9 1999/03/12 20:16:22 aitkaci Exp $"
        date: "$Date: 1999/03/12 20:16:22 $"
        revision: "$Revision: 1.9 $"

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: not destroyed
```

```
            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.

    feature -- Basic operations

        propagate_background_color
                -- Propagate the current background color of the container
                -- to the children.
            require
                exists: not destroyed

        propagate_foreground_color
                -- Propagate the current foreground color of the container
                -- to the children.
            require
                exists: not destroyed

    feature -- Implementation

        implementation: EV_CONTAINER_I

    end -- class EV_CONTAINER
```

3.1 EV_WINDOW

Basis for almost every application, the window is a basic GUI component that consists of a bordered rectangular area visible on the screen.

Any widget, except for a window, can be put inside this container. A window is a manager container that allows only one child.

Class

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 1.28 1999/03/12 20:16:29 aitkaci Exp $"
date: "$Date: 1999/03/12 20:16:29 $"
revision: "$Revision: 1.28 $"

class interface

EV_WINDOW

creation

make_top_level,
make

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: ***not*** *destroyed*

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

```
        require
            exists: not destroyed

    icon_pixmap: EV_PIXMAP
            -- Bitmap that could be used by the window manager
            -- as the application's icon
        require
            exists: not destroyed
        ensure
            valid_result: Result /= void

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

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

    parent: EV_WINDOW
            -- The parent of the Current window: a window
            -- If the widget is an EV_WINDOW without parent,
            -- this attribute will be Void

    title: STRING
            -- Application name to be displayed by
            -- the window manager
        require
            exists: not destroyed

    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: not destroyed

feature -- Status report

    is_iconic_state: BOOLEAN
            -- Does application start in iconic state?
        require
            exists: not destroyed

feature -- Status setting

    allow_resize
            -- Allow the resize of the window.
        require
            exists: not destroyed

    forbid_resize
            -- Forbid the resize of the window.
        require
            exists: not destroyed

    set_iconic_state
            -- Set start state of the application
            -- to be iconic.
        require
            exists: not destroyed

    set_maximize_state
            -- Set start state of the application to be
            -- maximized.
        require
            exists: not destroyed

    set_modal
            -- Make the window modal
        require
            exists: not destroyed

    set_normal_state
```

```
            -- Set start state of the application to be normal.
        require
            exists: not destroyed

    show
            -- Make the window visible on the screen
        require else
            exists: not destroyed

feature -- Element change

    set_icon_mask (pixmap: EV_PIXMAP)
            -- Make pixmap the new icon mask.
        require
            exists: not destroyed;
            valid_mask: is_valid (pixmap)

    set_icon_name (txt: STRING)
            -- Make txt the new icon name.
        require
            exists: not destroyed;
            valid_name: txt /= void

    set_icon_pixmap (pixmap: EV_PIXMAP)
            -- Make pixmap the new icon pixmap.
        require
            exists: not destroyed;
            valid_pixmap: is_valid (pixmap)

    set_maximum_height (value: INTEGER)
            -- Make value the new maximum_height.
        require
            exists: not destroyed;
            large_enough: value >= 0
        ensure
            maximum_height_set: maximum_height = value

    set_maximum_width (value: INTEGER)
            -- Make value the new maximum_width.
        require
            exists: not destroyed;
            large_enough: value >= 0
```

```
            ensure
                maximum_width_set: maximum_width = value

        set_title (txt: STRING)
                -- Make text the new title.
            require
                exists: not destroyed;
                valid_title: txt /= void

        set_widget_group (widget: EV_WIDGET)
                -- Make Current part of the group of widget.
            require
                exists: not destroyed;
                valid_widget: is_valid (widget)

    feature -- Implementation

        implementation: EV_WINDOW_I
                -- Implementation of window
                -- Depth_is_zero: depth = 0
                -- Has_no_parent: parent = Void

    feature -- Event - command association

        add_close_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
                -- Add cmd to the list of commands to be executed
                -- when the window is closed.
            require
                exists: not destroyed;
                valid_command: cmd /= void

        add_move_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
                -- Add cmd to the list of commands to be executed
                -- when the widget is moved.
            require
                exists: not destroyed;
                valid_command: cmd /= void

        add_resize_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
                -- Add cmd to the list of commands to be executed
                -- when the window is resized.
            require
```

```
            exists: not destroyed;
            valid_command: cmd /= void

feature -- Event -- removing command association

    remove_close_commands
            -- Empty the list of commands to be executed
            -- when the window is closed.
        require
            exists: not destroyed

    remove_move_commands
            -- Empty the list of commands to be executed
            -- when the widget is resized.
        require
            exists: not destroyed

    remove_resize_commands
            -- Empty the list of commands to be executed
            -- when the window is resized.
        require
            exists: not destroyed

end -- class EV_WINDOW
```

3.2 EV_DIALOG

Dialog is a special window that you can use for pop-up messages to the user and other similar tasks.

Dialog contains two areas to which you can add widgets:

- action area — at the bottom of the dialog. It is usually the area where the buttons are added.
- display area — on the top of the window, above the action area.

Class

```
indexing
    description: "EiffelVision dialog. A dialog is a window with predefine containers and widgets : a vertical box inside and a panel of button in the action-area (horizontal_box)."
    status: "See notice at end of class"
    id: "$Id: ev_dialog.e,v 1.8 1999/03/12 20:16:23 aitkaci Exp $"
    date: "$Date: 1999/03/12 20:16:23 $"
    revision: "$Revision: 1.8 $"
```

class interface
EV_DIALOG

creation
make

feature *-- Access*

action_area: EV_HORIZONTAL_BOX
-- The action area on the bottom of the window

display_area: EV_VERTICAL_BOX
-- The display area on the top of the window

end *-- class EV_DIALOG*

3.3 EV_FIXED

Fixed is an invisible, non-manager container that can contain an unlimited number of other widgets. The location of widgets inside a fixed widget is specified by the coordinates (widget attributes *x* and *y)* relative to the top left corner of the fixed widget.

Class

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 1.7 1999/03/12 20:16:24 aitkaci Exp $"
date: "$Date: 1999/03/12 20:16:24 $"
revision: "$Revision: 1.7 $"

class interface
EV_FIXED

creation
make

feature *-- Access*

manager: BOOLEAN

end *-- class EV_FIXED*

3.4 EV_BOX

A box is a manager that can receive an unlimited number of children, while class **EV_BOX** is a deferred ancestor of both **EV_VERTICAL_BOX** and **EV_HORIZONTAL_BOX**.

By default a box is *homogeneous* which means that parent gives equal amount of space to each child. The actual value is determined by the size of the largest child in the container.

Box can be set to *non-homogeneous* by using the feature *set_homogeneous* with a parameter *False*. If the box is *non-homogeneous*, each child has a space relative to the minimum size of itself.

If a child has attribute *expand* set to *False*, the container cannot allocate any extra space to the children. The remaining space is distributed among the other children.

You can add a space between each child. This space is called spacing. By default, the spacing is set to 0. You can change this value using the feature *set_spacing*.

Class

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 1.11 1999/02/09 01:09:00 aitkaci Exp $"
date: "$Date: 1999/02/09 01:09:00 $"
revision: "$Revision: 1.11 $"

deferred class interface

EV_BOX

feature *-- Access*

border_width: INTEGER
-- Border width around container
require
exists: ***not*** *destroyed*
ensure
positive_result: Result >= 0

feature *-- Element change (box specific)*

```
        set_border_width (value: INTEGER)
                -- Make value the new border width.
            require
                exist: not destroyed;
                positive_value: value >= 0
            ensure
                border_set: border_width = value

        set_homogeneous (flag: BOOLEAN)
                -- Homogenous controls whether each object in
                -- the box has the same size.
            require
                exist: not destroyed

        set_spacing (value: INTEGER)
                -- Spacing between the objects in the box
            require
                exist: not destroyed;
                positive_value: value >= 0

end -- class EV_BOX
```

3.5 EV_VERTICAL_BOX

A vertical box packs the children in one column.

Class

```
indexing
    description: "EiffelVision vertical box."
    status: "See notice at end of class"
    id: "$Id: ev_vertical_box.e,v 1.5 1999/03/12 20:16:28 aitkaci Exp $"
    date: "$Date: 1999/03/12 20:16:28 $"
    revision: "$Revision: 1.5 $"

class interface
    EV_VERTICAL_BOX
```

creation

make

end *-- class EV_VERTICAL_BOX*

3.6 EV_HORIZONTAL_BOX

An horizontal box packs the children in one row.

Class

indexing

description: "EiffelVision horizontal box."
status: "See notice at end of class"
id: "$Id: ev_horizontal_box.e,v 1.6 1999/03/12 20:16:25 aitkaci Exp $"
date: "$Date: 1999/03/12 20:16:25 $"
revision: "$Revision: 1.6 $"

class interface

EV_HORIZONTAL_BOX

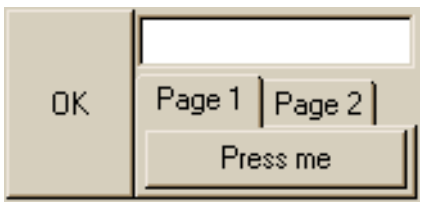
creation

make

end *-- class EV_HORIZONTAL_BOX*

3.7 EV_TABLE

A table is a manager container that can receive an unlimited number of children. A table contains a grid of rows and columns where you can place the widgets. The widgets can occupy as many spaces in the table as you want.



The *homogeneous* attribute of the table reffers to the size of the individual grid square. If *homogeneous* is set to *True*, the grid square resize to the size of the largest

widget in the table. If *homogeneous* is set to *False*, the grid square size is determined by the tallest widget in the same row, and the widest widget in the same column.

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 3.7.1.

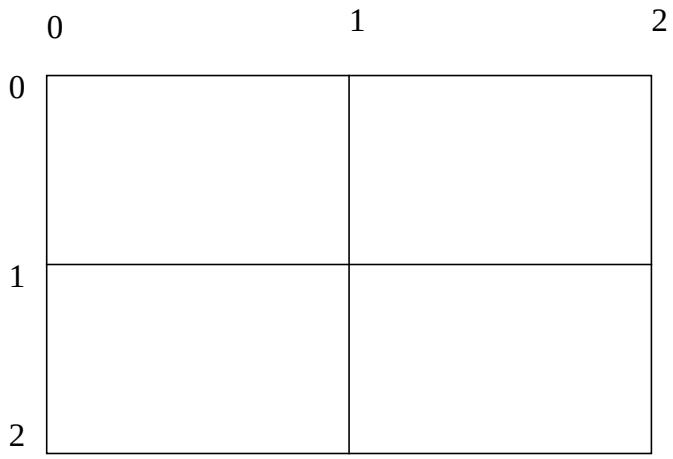


Figure 3.7.1 Table layout with two rows and two columns

The coordinate system starts in the upper left hand corner.

Class

indexing

description: "EiffelVision table. Invisible container that allows unlimited number of other widgets to be packed inside it. A table controls the children%'s location and size automatically."
status: "See notice at end of class"
id: "$Id: ev_table.e,v 1.4 1999/03/12 20:16:27 aitkaci Exp $"
date: "$Date: 1999/03/12 20:16:27 $"
revision: "$Revision: 1.4 $"

class interface

EV_TABLE

creation

make

feature *-- Status report*

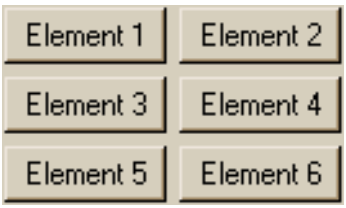
columns: INTEGER
-- Number of columns
require
exists: ***not*** *destroyed*

```
        rows: INTEGER
                -- Number of rows
            require
                exists: not destroyed

feature -- Status settings

        set_child_position (the_child: EV_WIDGET; top, left, bottom, right: INTEGER)
                -- Set the position and the size of the given child in
                -- the table. top, left, bottom and right give the
                -- zero-based coordinates of the child in the grid.
                --
                -- 0 1 2
                -- 0 +----------+---------+
                -- | | |
                -- 1 +----------+---------+
                -- | | |
                -- 2 +----------+---------+
                --
                -- This feature must be called after the creation of
                -- the child, otherwise, the child won't appear in
                -- the table.
            require
                exists: not destroyed;
                the_child_not_void: the_child /= void;
                bottom_larger_than_top: bottom > top;
                right_larger_than_left: right > left

        set_column_spacing (value: INTEGER)
                -- Spacing between two columns of the table
            require
                exist: not destroyed;
                positive_value: value >= 0

        set_homogeneous (flag: BOOLEAN)
                -- Homogenous controls whether each object in
                -- the box has the same size.
            require
                exist: not destroyed

        set_row_spacing (value: INTEGER)
```

```
        -- Spacing between two rows of the table
    require
        exist: not destroyed;
        positive_value: value >= 0

end -- class EV_TABLE
```

3.8 EV_DYNAMIC_TABLE

A dynamic table is a table where all the children occupied only one cell and where the placement of the children is done automaticaly. You can choose both the way the table positions the children (horizontaly or verticaly) and the width of the table. For example, in the following example, the table has a *row_layout* with a *finite_dimension* set to 2.



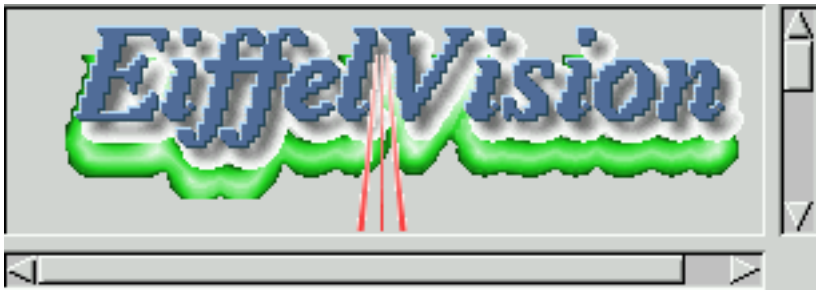
Class

```
indexing
    description: " EiffelVision dynamic table. Invisible container that allows unlimited number of other
widgets to be packed inside it. A dynamic table controls the children%'s location and size
automatically."
    note: " In this table, each child fill one cell. The user choose the way to lay the children out. If the
children are laid in rows, the number of colums must be finite and the one of rows is infinite; if they area
laid out in columns, it%'s the contrary."
    note2: " By default, a dynamic table is the equivalent of an horizonatl box."
    status: "See notice at end of class"
    id: "$Id: ev_dynamic_table.e,v 1.3 1999/03/12 20:16:23 aitkaci Exp $"
    date: "$Date: 1999/03/12 20:16:23 $"
    revision: "$Revision: 1.3 $"

class interface
    EV_DYNAMIC_TABLE

creation
    make

feature -- Status report
```

```
        is_row_layout: BOOLEAN
                -- Are children laid out in rows?
                -- False by default
            require
                exists: not destroyed

feature -- Status setting

        set_column_layout
                -- Lay the children out in columns.
            require
                exists: not destroyed
            ensure
                column_layout: not is_row_layout

        set_finite_dimension (a_number: INTEGER)
                -- Set number of columns if row
                -- layout, or number of row if column
                -- layout.
            require
                exists: not destroyed;
                positive_number: a_number > 0

        set_row_layout
                -- Lay the children out in rows.
            require
                exists: not destroyed
            ensure
                row_layout: is_row_layout

end -- class EV_DYNAMIC_TABLE
```

3.9 EV_SCROLLABLE_AREA

A scrollable area is a *non-manager* container that can receive only one child and displays as an area surrounded by horizontal and vertical scrollbars.



If the inside widget is bigger than the visible area, you can use the scrollbars to scroll through the widget. The size of the scroll arrow corresponds to the visible size of the widget (the size of the scrollable area), while the size of the scrollbar corresponds to the size of the whole widget.

Class

indexing

description: "EiffelVision scrollable area. Scrollable area is a container with scrollbars. Scrollable area offers automatic scrolling for its child."

status: "See notice at end of class"

id: "$Id: ev_scrollable_area.e,v 1.6 1999/03/12 20:16:27 aitkaci Exp $"

date: "$Date: 1999/03/12 20:16:27 $"

revision: "$Revision: 1.6 $"

class interface

EV_SCROLLABLE_AREA

creation

make

feature *-- Access*

manager: BOOLEAN

end *-- class EV_SCROLLABLE_AREA*

3.10 EV_FRAME

A frame is a *manager* container that can receive only one child and displays as an area surrounded by a rectangle. You can add a title to a frame which appears in the top left corner of the rectangle.

Class

indexing

description: "EiffelVision frame. A frame is a container with a line around it. A label can be set on this line or not."

status: "See notice at end of class"

id: "$Id: ev_frame.e,v 1.3 1999/03/12 20:18:42 aitkaci Exp $"

date: "$Date: 1999/03/12 20:18:42 $"

revision: "$Revision: 1.3 $"

class interface

EV_FRAME

creation

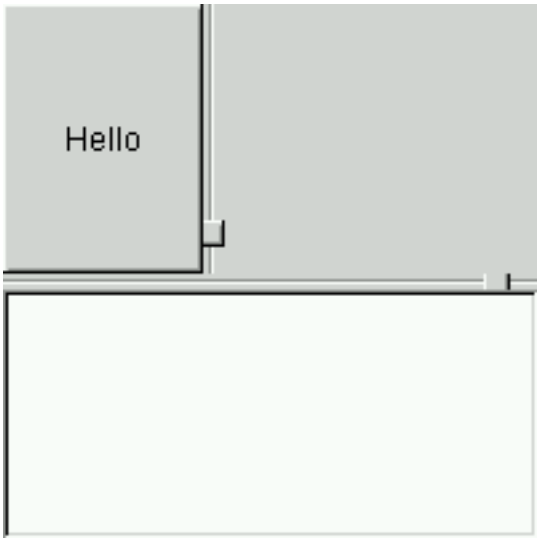
make,

make_with_text

end *-- class EV_FRAME*

3.11 EV_SPLIT_AREA

A split area is a manager container that can receive only two children, which are separated by a groove.



You can control the relative size of the two children by physically moving the groove.

EV_SPLIT_AREA is a deferred ancestor of both **EV_VERTICAL_SPLIT_AREA** and **EV_HORIZONTAL_SPLIT_AREA**.

Class

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 1.4 1999/01/08 21:24:36 aitkaci Exp $"

date: "$Date: 1999/01/08 21:24:36 $"

revision: "$Revision: 1.4 $"

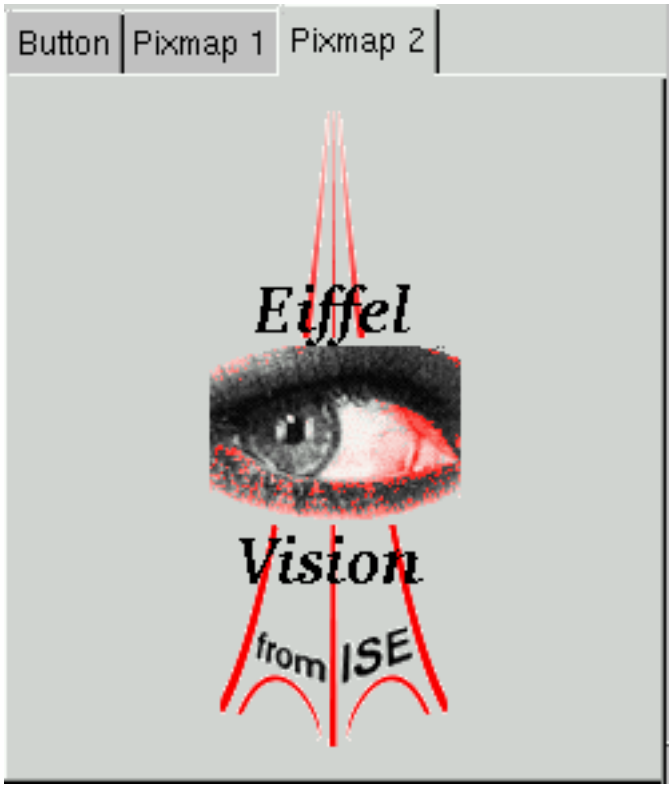
deferred class interface

EV_SPLIT_AREA

end *-- class EV_SPLIT_AREA*

3.12 EV_NOTEBOOK

A Notebook is a collection of pages that overlap each other, with a tab corresponding to each page although only one of the pages is visible.



The tabs display along the are top, bottom, left or right edge of the page. When you click a tab, the corresponding page is made visible.

If there are a lot of tabs, it may not be possible to display all tabs. In this case, you can set the number of visible tabs. If there are more tabs than visible tabs, scroll buttons that you can use to control which of the tabs appear, display.

A notebook is a manager container that can receive an unlimited number of children. However each page can receive only one child. Pages can be added to and removed from the Notebook.

Class

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 1.10 1999/03/12 20:16:26 aitkaci Exp $"

date: "$Date: 1999/03/12 20:16:26 $"

revision: "$Revision: 1.10 $"

class interface

EV_NOTEBOOK

creation

make

feature *-- Status report*

count: INTEGER
-- Number of pages in the notebook

current_page: INTEGER
-- Index of the page currently opened

feature *-- Status setting*

set_current_page (index: INTEGER)
-- Make the index-th page the currently opened page.
require
exists: ***not*** *destroyed;*
valid_index: index >= 1 ***and*** *index <= count*

set_tab_bottom
-- Put the tabs at the bottom of the notebook.
require
exists: ***not*** *destroyed*

set_tab_left
-- Put the tabs at the left of the notebook.
require
exists: ***not*** *destroyed*

set_tab_right
-- Put the tabs at the right of the notebook.
require
exists: ***not*** *destroyed*

set_tab_top
-- Put the tabs at the top of the notebook.
-- default.
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

    set_page_title (index: INTEGER; str: STRING)
            -- Set the label of the index page of the notebook.
            -- The first page is the page number 1.
        require
            exists: not destroyed;
            good_index: index <= count

feature -- Event - command association

    add_switch_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add 'cmd' to the list of commands to be executed
            -- the a page is switch in the notebook.
        require
            exists: not destroyed;
            valid_command: cmd /= void

feature -- Event -- removing command association

    remove_switch_commands
            -- Empty the list of commands to be executed
            -- when a page is switch in the notebook.
        require
            exists: not destroyed

end -- class EV_NOTEBOOK
```

4

Primitives

A primitive is a childless widget — other widgets cannot be placed inside it. Nevertheless, some primitives can contain specific items as the **EV_LIST** or **EV_TREE**.

indexing

description: "EiffelVision primitive. Deferred class, ancestor of many widgets"
status: "See notice at end of class"
id: "$Id: ev_primitive.e,v 1.4 1999/03/12 20:18:47 aitkaci Exp $"
date: "$Date: 1999/03/12 20:18:47 $"
revision: "$Revision: 1.4 $"

deferred class interface

EV_PRIMITIVE

end *-- class EV_PRIMITIVE*

4.1 EV_BUTTON

Class **EV_BUTTON** is one of the most useful primitive. It is also a common ancestor for different button classes.

A button acquires a 3D appearance as it is implemented by the underlying toolkit.

A button can contain a text, a pixmap, or both. When both are present, there are two different presentation methods:

- pixmap on the top, label on the bottom,
- pixmap on the left and label on the right.

Class

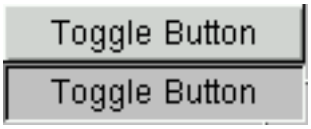
```
indexing
        description: "EiffelVision button. Basic GUI push button. This is also a base class for other buttons classes"
        status: "See notice at end of class"
        id: "$Id: ev_button.e,v 1.16 1999/03/04 00:35:15 aitkaci Exp $"
        date: "$Date: 1999/03/04 00:35:15 $"
        revision: "$Revision: 1.16 $"

class interface
        EV_BUTTON

creation
        make,
        make_with_text

feature -- Event - command association

        add_click_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
                        -- Add 'cmd' to the list of commands to be executed
                        -- the button is pressed.
                require
                        exists: not destroyed;
                        valid_command: cmd /= void

feature -- Event -- removing command association

        remove_click_commands
                        -- Empty the list of commands to be executed when
                        -- the button is pressed.
                require
                        exists: not destroyed

end -- class EV_BUTTON
```

4.2 EV_TOGGLE_BUTTON

EV_TOGGLE_BUTTON is a descendant of **EV_BUTTON** and is very similar, except that it is always in one of two states, alternated by a click :

- *deselected* — then you can click on it and it will pop down,

• *selected* — then you can clicke on it again and it will pop back up.



Toggle buttons provide the basis for check buttons and radio buttons. Consequently, radio and check buttons inherit many of the calls used for toggle buttons.

The default state after creation is *deselected*.

Class

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 1.14 1999/03/12 20:18:50 aitkaci Exp $"

date: "$Date: 1999/03/12 20:18:50 $"

revision: "$Revision: 1.14 $"

class interface

EV_TOGGLE_BUTTON

creation

make,

make_with_text

feature *-- Status report*

state: BOOLEAN

-- Is toggle pressed.

require

exists: ***not*** *destroyed*

feature *-- Status setting*

set_state (flag: BOOLEAN)

-- Set Current toggle on and set

-- pressed to True.

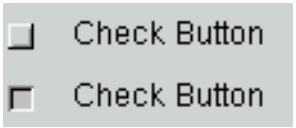
require

exists: ***not*** *destroyed*

```
            ensure
                correct_state: state = flag

        toggle
                -- Change the state of the toggel button to
                -- opposite
            require
                exists: not destroyed
            ensure
                state_is_true: state = not old state

feature -- Event - command association

        add_toggle_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
                -- Add 'cmd' to the list of commands to be executed
                -- when the button is toggled.
            require
                exists: not destroyed;
                valid_command: cmd /= void

feature -- Event -- removing command association

        remove_toggle_commands
                -- Empty the list of commands to be executed
                -- when the button is toggled.
            require
                exists: not destroyed

end -- class EV_TOGGLE_BUTTON
```

4.3 EV_CHECK_BUTTON

Check buttons are similar to toggle buttons. Rather than being buttons with a label and/or a pixmap inside them, they look like check buttons on the underlying toolkit that contains usually a name preceded by a check mark or bullet that turns on or off each time the user click on it.



Class

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 1.8 1999/03/26 04:53:28 aitkaci Exp $"
date: "$Date: 1999/03/26 04:53:28 $"
revision: "$Revision: 1.8 $"

class interface

EV_CHECK_BUTTON

creation

make,
make_with_text

end *-- class EV_CHECK_BUTTON*

4.4 EV_RADIO_BUTTON

Radio buttons are similar to check buttons except that radio buttons are grouped and only one item can be selected at a time.

Radio buttons that have the same parent belong to the same group. However, it is possible to have several radio button groups inside the same parent : for example, a window. This is not a problem since you can use special containers to group radio buttons. For example, a vertical-box inside an **EV_FRAME** is a good component to group radio buttons, because it also groups the buttons visually inside a border.

Class

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 1.6 1999/03/26 04:53:50 aitkaci Exp $"
date: "$Date: 1999/03/26 04:53:50 $"
revision: "$Revision: 1.6 $"

class interface
EV_RADIO_BUTTON

creation
make,
make_with_text

feature *-- Initialization*

make (par: EV_CONTAINER)
-- radio button with par as parent.

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

end *-- class EV_RADIO_BUTTON*

4.5 EV_OPTION_BUTTON

EV_OPTION_BUTTON is a descendant of **EV_BUTTON**. When you click an option button, a pop-op menu appears, and the item you select displays as the text of the button.

Only one menu can be added in an option button at a time.

Class

indexing
description: "EiffelVision option button is a button that displays a popup_menu when we click on it."
id: "$Id: ev_option_button.e,v 1.5 1999/03/26 01:35:35 aitkaci Exp $"
date: "$Date: 1999/03/26 01:35:35 $"
revision: "$Revision: 1.5 $"

class interface
EV_OPTION_BUTTON

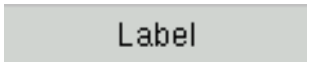
creation

make

end *-- class EV_OPTION_BUTTON*

4.6 EV_LABEL

A label is a static text that you can place anywhere in a window. For example, as an explanation next to a text field.



Class

indexing

description: "EiffelVision label"
status: "See notice at end of class"
id: "$Id: ev_label.e,v 1.13 1999/03/12 20:18:43 aitkaci Exp $"
date: "$Date: 1999/03/12 20:18:43 $"
revision: "$Revision: 1.13 $"

class interface

EV_LABEL

creation

make,
make_with_text

end *-- class EV_LABEL*

4.7 EV_TEXT_COMPONENT

EV_TEXT_COMPONENT is a deferred ancestor of the **EV_TEXT_AREA** and **EV_TEXT_FIELD**. It contains several text management tools, including cutting, copying or pasting text.

Class

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 1.8 1999/03/22 16:37:10 aitkaci Exp $"

```
        date: "$Date: 1999/03/22 16:37:10 $"
        revision: "$Revision: 1.8 $"

deferred class interface
        EV_TEXT_COMPONENT

feature -- Access

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

        text_length: INTEGER
                -- Length of the text in the widget
            require
                exists: not destroyed

feature -- Status report

        has_selection: BOOLEAN
                -- Is something selected?
            require
                exist: not destroyed

        position: INTEGER
                -- Current position of the caret.
            require
                exist: not destroyed

        selection_end: INTEGER
                -- Index of the last character selected
            require
                exist: not destroyed;
                has_selection: has_selection
            ensure
                result_large_enough: Result >= 0;
                result_small_enough: Result <= text_length

        selection_start: INTEGER
                -- Index of the first character selected
            require
```

```
            exist: not destroyed;
            has_selection: has_selection
        ensure
            result_large_enough: Result >= 0;
            result_small_enough: Result <= text_length

feature -- Status setting

    set_editable (flag: BOOLEAN)
            -- flag true make the component read-write and
            -- flag false make the component read-only.
        require
            exists: not destroyed

    set_maximum_text_length (value: INTEGER)
            -- Make value the new maximal lenght of the text
            -- in characte number.
        require
            exist: not destroyed;
            valid_length: value >= 0

    set_position (pos: INTEGER)
            -- Set current insertion position.
        require
            exist: not destroyed;
            valid_pos: pos > 0 and pos <= text_length

feature -- Element change

    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

    set_text (txt: STRING)
```

```
            -- Make txt the new text.
        require
            exists: not destroyed;
            not_void: txt /= void
        ensure
            text_set: text.is_equal (txt)

feature -- Resizing

    set_minimum_width_in_characters (nb: INTEGER)
            -- Make nb characters visible on one line.
        require
            exists: not destroyed;
            valid_nb: nb > 0

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: not destroyed;
            has_selection: has_selection

    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: not destroyed;
            has_selection: has_selection

    delete_selection
            -- Delete the current selection.
        require
            exist: not destroyed;
            has_selection: has_selection
        ensure
```

```
            has_no_selection: not has_selection

    deselect_all
            -- Unselect the current selection.
        require
            exist: not destroyed;
            has_selection: has_selection
        ensure
            has_no_selection: not has_selection

    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: not destroyed

    select_all
            -- Select all the text.
        require
            exist: not destroyed;
            positive_length: text_length > 0
        ensure
            has_selection: has_selection;
            selection_start_set: selection_start = 0;
            selection_end_set: selection_end <= text_length + 2

    select_region (start_pos, end_pos: INTEGER)
            -- Select (hilight) the text between
            -- start_pos and end_pos
        require
            exist: not destroyed;
            valid_start: start_pos > 0 and start_pos <= text_length;
            valid_end: end_pos > 0 and end_pos <= text_length
        ensure
            has_selection: has_selection;
            selection_start_set: selection_start = start_pos;
            selection_end_set: selection_end = end_pos

feature -- Event - command association
```

```
    add_change_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add 'cmd' to the list of commands to be executed
            -- when the text of the widget have changed.
        require
            exists: not destroyed;
            valid_command: cmd /= void

feature -- Event -- removing command association

    remove_change_commands
            -- Empty the list of commands to be executed
            -- when the text of the widget have changed.
        require
            exists: not destroyed

end -- class EV_TEXT_COMPONENT
```

4.8 EV_TEXT_FIELD

A text field accepts a single line of enterd text.

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. You can choose the maximum length of the text accepted.

Sometimes it is necessary to check the validity of entered text. The following is a suggestion for the validity checking:

Offer a class **EV_TEXT_FILTER** with a redefineable feature *filter (char: CHARACTER): BOOLEAN*. *filter* returns *True*, if the character is valid. Another creation procedure for **EV_TEXT_FIELD** also needs to be added: *make_with_filter (filter: EV_TEXT_FILTER)*.This is not yet very effective. A more efficient solution is to create the filter object, using a regular expression to describe the validity of the input

Class

```
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 1.12 1999/03/22 16:55:45 aitkaci Exp $"
    date: "$Date: 1999/03/22 16:55:45 $"
    revision: "$Revision: 1.12 $"
```

class interface
EV_TEXT_FIELD

creation
make,
make_with_text

feature *-- Event - command association*

add_activate_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
-- Add 'cmd' to the list of commands to be executed
-- when the text field is activated, ie when the user
-- press the enter key.
require
exists: ***not*** *destroyed;*
valid_command: cmd /= void

feature *-- Event -- removing command association*

remove_activate_commands
-- Empty the list of commands to be executed
-- when the text field is activated, ie when the user
-- press the enter key.
require
exists: ***not*** *destroyed*

end *-- class EV_TEXT_FIELD*

4.9 EV_PASSWORD_FIELD

A password field is a text field that ask for a password in an application. The text typed into the password entry does not display, rather a certain character substitutes for the typed character. An asterix (*) is the substituted characher, by default.

Class

indexing
description: "EiffelVision password field. A text field That displays always the same character."

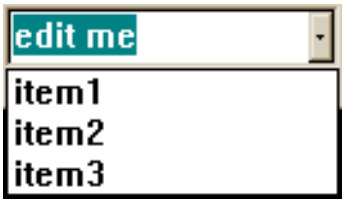
```
        status: "See notice at end of class"
        date: "$Date: 1999/02/18 23:30:46 $"
        revision: "$Revision: 1.2 $"

class interface
        EV_PASSWORD_FIELD

creation
        make

feature -- Access

        character: CHARACTER
                -- Displayed character instead of the text.
            require
                exists: not destroyed

feature -- Element change

        set_character (char: CHARACTER)
                -- Make char the new character displayed in the
                -- password field.
            require
                exists: not destroyed

end -- class EV_PASSWORD_FIELD
```

4.10 EV_SPINBUTTON

Spinbuttons are single line entries that have two small arrow buttons along the right edge of the text field. Spinbutton can only contain numeric values. When you click the arrow buttons, the value of the entry decreases or increases, depending on the button you click.

4.11 EV_COMBO_BOX

A combo box contains a text field, a button and a list.



When you click the button, a list of choices displays. You can either type a value in the text field or select one from the list. A combo-box can be editable or not. When it is not editable, you can click any where in the combo-box to show the list.

Class

indexing

description: "EiffelVision Combo-box. A combo-box contains a text field and a button. When the button is pressed, a list of possible choices is opened."

note: "The 'height%' of a combo-box is the one of the text field. To have the height of the text field plus the list, use extended_height."

status: "See notice at end of class"

names: widget

date: "$Date: 1999/01/27 00:44:26 $"

revision: "$Revision: 1.14 $"

class interface

EV_COMBO_BOX

creation

make

feature *-- Measurement*

extended_height: INTEGER

-- height of the combo-box when the children are

-- visible.

require

exists: ***not*** *destroyed*

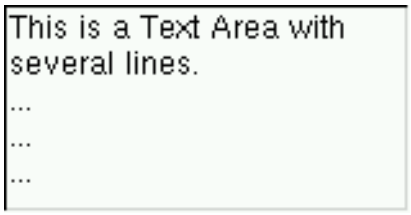
feature *-- Implementation*

implementation: EV_COMBO_BOX_I

end *-- class EV_COMBO_BOX*

4.12 EV_TEXT_AREA

A text area is a text field that accepts multiple lines. Therefore, all the words have the same font, size and color.



The property maximum length controls the number of characters in the entire text. A text area has two creation routines : with or without scrollbars.

Class

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 1.2 1999/01/08 21:23:26 aitkaci Exp $"
date: "$Date: 1999/01/08 21:23:26 $"
revision: "$Revision: 1.2 $"

class interface

EV_TEXT_AREA

creation

make

end *-- class EV_TEXT_AREA*

4.13 EV_TEXT_EDITOR

A text editor is a complete multi-line text widget that contains editing features including multiple colos and fonts for the text.

4.14 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. The relevant features are *set_double_dashed_line*, *set_double_line*, *set_no_line*, *set_single_dashed_line*, and *set_single_line***.**

EV_SEPARATOR is a deferred of both **EV_VERTICAL_SEPARATOR** and **EV_HORIZONTAL_SEPARATOR**.

Class

indexing

description: "EiffelVision separator."
status: "See notice at end of class"
date: "$Date: 1999/01/27 00:46:06 $"
revision: "$Revision: 1.3 $"

deferred class interface

EV_SEPARATOR

end *-- class EV_SEPARATOR*

4.15 EV_RANGE

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

4.16 EV_SCROLLBAR

A scrollbar is a simple concept. It contains a scroll box that indicates the relative position within the scrollable material (or position within the scrollbar) and scroll arrows at either end for movement. You can drag the scroll box to a new position, click a scroll arrow to movethe scroll box on line unit or click in the scroll bar to move the scroll box one page unit. You can set the line and page unit.

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

EV_SCROLLBAR is a deferred ancestor of both **EV_HORIZONTAL_SCROLLBAR** and **EV_VERTICAL_SCROLLBAR**.

Events that can occur on a scrollbar include the movement of the scroll box and the position being changed.

Depending on the toolkit, it is possible to increase the scroll box speed. Speed is set by an *initial_delay* and a *repeat_delay* (*set_inital_delay* and *set_repeat_delay*). Also affecting movement is the *granularity,* which affects how much the scroll bar will move as well as the *maximum* and *minimum* values for the range of movement. The routines to set the movement values are *set_granularity, set_maximum* and *set_minimum*.

4.17 EV_SCALE

A scale can be considered as a scrollbar with a label. Unlike a scrollbar, the value of the label on the scale represents a numeric value you set.

EV_SCALE is a deferred ancestor of both **EV_HORIZONTAL_SCALE** and **EV_VERTICAL_SCALE**.

Like a scrollbar, a scale has a *move* event and a *value_changed* event. The *granularity*, *minimum*, *maximum*, *scroll box* and *orientation* values 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. You can set the scale to only output values (*set_output_only*) and then query (*is_output_only*). The label display using the *set_text* feature and queried using the *text* feature. The numerical value of the scale displays by setting *is_value_shown* through the *show_value* feature.

By default, the maximum of the scale is the lower-left corner for the vertical scales and the lower-right corner for the horizontal ones. However, you can change the default behavior by using *set_maximum_right_bottom* and query the value using *is_maximum_right_bottom.*

4.18 EV_LIST

A list contains a selectable list of options. You can allow one or more selections. Inside the list, you can add some **EV_LIST_ITEM.**

Class

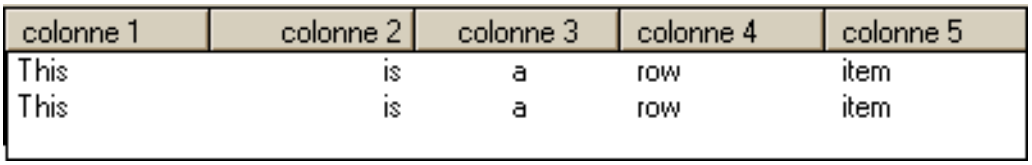
```
indexing
        description: "EiffelVision list. Contains a list of items from which the user can select."
        status: "See notice at end of class"
        id: "$$"
        date: "$Date: 1999/03/04 00:35:16 $"
        revision: "$Revision: 1.15 $"

class interface
        EV_LIST

creation
        make

feature -- Access

        count: INTEGER
                        -- Number of rows
                require
                        exists: not destroyed

        get_item (index: INTEGER): EV_LIST_ITEM
                        -- Give the item of the list at the zero-base
                        -- index.
                require
```

```
                exists: not destroyed;
                item_exists: index <= count

        selected_item: EV_LIST_ITEM
                -- Item which is currently selected
                -- It needs to be in single selection mode
            require
                exists: not destroyed;
                single_selection: not is_multiple_selection

        selected_items: LINKED_LIST [EV_LIST_ITEM]
                -- List of all the selected items. For a single
                -- selection list, it gives a list with only one
                -- element which is selected_item. Therefore, one
                -- should use selected_item rather than
                -- selected_items for a single selection list
            require
                exists: not destroyed

feature -- Status report

        is_multiple_selection: BOOLEAN
                -- True if the user can choose several items
                -- False otherwise
            require
                exist: not destroyed

        selected: BOOLEAN
                -- Is at least one item selected ?
            require
                exists: not destroyed

feature -- Status setting

        select_item (index: INTEGER)
                -- Select an item at the one-based index the list.
            require
                exists: not destroyed;
                index_large_enough: index > 0;
                index_small_enough: index <= count

        set_multiple_selection
```

```
            -- Allow the user to do a multiple selection simply
            -- by clicking on several choices.
        require
            exists: not destroyed

    set_single_selection
            -- Allow the user to do only one selection. It is the
            -- default status of the list
        require
            exists: not destroyed

feature -- Element change

    clear_items
            -- Clear all the items of the list.
        require
            exists: not destroyed

feature -- Event -- removing command association

    remove_selection_commands
            -- Empty the list of commands to be executed
            -- when the selection has changed.
        require
            exists: not destroyed

feature -- Event : command association

    add_selection_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed
            -- when the selection has changed.
        require
            exists: not destroyed;
            valid_command: cmd /= void

end -- class EV_LIST
```

4.19 EV_MULTI_COLUMN_LIST

A multi-column list contains the functionalities of a list with the difference that its components are of type **EV_MULTI_COLUMN_LIST_ROW**. A multi-column list row consists of several parts, with each part in the list represented by the item in one column. A multi-column list also has a title row, which displays above the list.



colonne 1	colonne 2	colonne 3	colonne 4	colonne 5
This	is	a	row	item
This	is	a	row	item

The title row controls which columns are visible and the size of the columns.

Class

indexing

description: "EiffelVision multi-column-list. Contains a list of items from which the user can select."

note: "The list start at the index 1, the titles are not count amongthe rows. The columns start also at the index 1."

status: "See notice at end of class"

id: "$$"

date: "$Date: 1999/03/04 00:35:16 $"

revision: "$Revision: 1.11 $"

class interface

EV_MULTI_COLUMN_LIST

creation

make_with_size

***feature** -- Access*

columns: INTEGER

-- Number of columns in the list.

require

*exists: **not** destroyed*

get_item (index: INTEGER): EV_MULTI_COLUMN_LIST_ROW

-- Give the item of the list at the one-base

-- index.

require

*exists: **not** destroyed;*

item_exists: index <= rows

rows: INTEGER

-- Number of rows

require

```
            exists: not destroyed

    selected_item: EV_MULTI_COLUMN_LIST_ROW
            -- Item which is currently selected, for a multiple
            -- selection, it gives the last selected item.
        require
            exists: not destroyed;
            single_selection: not is_multiple_selection

    selected_items: LINKED_LIST [EV_MULTI_COLUMN_LIST_ROW]
            -- List of all the selected items. For a single
            -- selection list, it gives a list with only one
            -- element which is selected_item. Therefore, one
            -- should use selected_item rather than
            -- selected_items for a single selection list
        require
            exists: not destroyed

feature -- Status report

    is_multiple_selection: BOOLEAN
            -- True if the user can choose several items
            -- False otherwise
        require
            exist: not destroyed

    selected: BOOLEAN
            -- Is at least one item selected ?
        require
            exists: not destroyed

feature -- Status setting

    hide_title_row
            -- Hide the row of the titles.
        require
            exists: not destroyed

    set_center_alignment (column: INTEGER)
            -- Align the text of the column at left.
            -- Cannot be used for the first column which is
            -- always left aligned.
```

```
            require
                exists: not destroyed;
                column_exists: column > 1 and column <= columns

        set_left_alignment (column: INTEGER)
                -- Align the text of the column at left.
                -- Cannot be used for the first column which is
                -- always left aligned.
            require
                exists: not destroyed;
                column_exists: column > 1 and column <= columns

        set_multiple_selection
                -- Allow the user to do a multiple selection simply
                -- by clicking on several choices.
            require
                exists: not destroyed

        set_right_alignment (column: INTEGER)
                -- Align the text of the column at left.
                -- Cannot be used for the first column which is
                -- always left aligned.
            require
                exists: not destroyed;
                column_exists: column > 1 and column <= columns

        set_single_selection
                -- Allow the user to do only one selection. It is the
                -- default status of the list
            require
                exists: not destroyed

        show_title_row
                -- Show the row of the titles.
            require
                exists: not destroyed

    feature -- Element change

        clear_items
                -- Clear all the items of the list.
            require
```

```
            exists: not destroyed

    set_column_title (txt: STRING; column: INTEGER)
            -- Make txt the title of the one-based column.
        require
            exists: not destroyed;
            column_exists: column >= 1 and column <= columns

    set_column_width (value: INTEGER; column: INTEGER)
            -- Make value the new width of the one-based column.
        require
            exists: not destroyed;
            column_exists: column >= 1 and column <= columns

    set_rows_height (value: INTEGER)
            -- Makevalue the new height of all the rows.
        require
            exists: not destroyed

feature -- Event -- removing command association

    remove_column_click_commands
            -- Empty the list of commands to be executed
            -- when a column is clicked.
        require
            exists: not destroyed

    remove_selection_commands
            -- Empty the list of commands to be executed
            -- when the selection has changed.
        require
            exists: not destroyed

feature -- Event : command association

    add_column_click_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed
            -- when a column is clicked.
        require
            exists: not destroyed;
            valid_command: cmd /= void
```

```
        add_selection_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
                -- Add cmd to the list of commands to be executed
                -- when the selection has changed.
            require
                exists: not destroyed;
                valid_command: cmd /= void

end -- class EV_MULTI_COLUMN_LIST
```

4.20 EV_TREE

A tree is a structure that represents data hierarchically. Each data item in a tree is of type **EV_TREE_ITEM**.

You can add a tree item directly to the tree, making it a root item, or to another item, that becomes a sub-tree item. A tree will have options that control the display : with lines, dashes, nothing...

Class

```
indexing
        description: "EiffelVision tree. A tree show a hierarchy with several levels of items."
        status: "See notice at end of class"
        id: "$$"
        date: "$Date: 1999/03/04 19:14:17 $"
        revision: "$Revision: 1.6 $"

class interface
        EV_TREE

creation
        make

feature -- Event -- removing command association

        remove_selection_commands
```

```
        -- Empty the list of commands to be executed
        -- when the selection has changed.
    require
        exists: not destroyed

feature -- Event : command association

    add_selection_command (a_command: EV_COMMAND; arguments: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed
        -- when an item is selected.
    require
        exists: not destroyed

end -- class EV_TREE
```

4.21 EV_PROGRESS_BAR

A progressbar is a control that displays the percentage of a particular process that has been complted, such as compilation. You can only set the level of the bar, you can not query the status.

EV_PROGRESS_BAR is a deferred ancestor of both **EV_VERTICAL_PROGRESS_BAR** and **EV_HORIZONTAL_PROGRESS_BAR**.

Class

4.22 EV_DRAWING_AREA

A drawing area is a widget on which you can draw. However, the area does not automatically refresh itself. You must use an *expose* event to redraw the drawing area when necessary.

If you want the drawing area to redraw itself automatically, you must attach a pixmap to the drawing area and draw on this pixmap. The pixmap is then copied to the drawing area automaticaly, when necessary.

5

Items

Items are specific elements that you can add to certain widgets. To add an item to a widget, you must make the widget the parent of the item. An item can have a parent Void. Each type of item corresponds to a particular type of widget that accepts them, for example, you can add an **EV_TREE_ITEM** only to an **EV_TREE**.

An Item contains uselly a label and a pixmap. An item will also have a data attribute that you can set.

In the current implementation, it has no effect to attach a pixmap to an item.

Class

indexing

description: "EiffelVision item. Top class of menu_item, list_item and tree_item. This item isn%'t a widget, because most of the features of the widgets are inapplicable here."
status: "See notice at end of class"
id: "$$"
date: "$Date: 1999/03/15 22:54:04 $"
revision: "$Revision: 1.11 $"

deferred class interface

EV_ITEM

feature *-- Access*

parent_widget: EV_WIDGET
-- Parent widget of the current item
require
exists: ***not*** *destroyed*

text: STRING
-- Current label of the item
require

```
            exists: not destroyed

feature -- Element change

    set_text (txt: STRING)
            -- Make txt the new label of the item.
        require
            exists: not destroyed;
            valid_text: txt /= void
        ensure
            text_set: text.is_equal (txt)

feature -- Implementation

    implementation: EV_ITEM_I

feature -- Event -- removing command association

    remove_activate_commands
            -- Empty the list of commands to be executed when
            -- the item is activated.
        require
            exists: not destroyed

    remove_deactivate_commands
            -- Empty the list of commands to be executed when
            -- the item is deactivated.
        require
            exists: not destroyed

feature -- Event : command association

    add_activate_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed
            -- when the item is activated.
        require
            exists: not destroyed;
            valid_command: cmd /= void

    add_deactivate_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed
            -- when the item is unactivated.
```

```
        require
            exists: not destroyed;
            valid_command: cmd /= void

end -- class EV_ITEM
```

5.1 EV_LIST_ITEM

The parent of a list item can be either an **EV_LIST** or an **EV_COMBO_BOX**.

In the future implementation, the pixmap should be visible.

```
indexing
    description: "EiffelVision list item. This items are used in the lists."
    status: "See notice at end of class"
    id: "$$"
    date: "$Date: 1999/03/15 22:55:23 $"
    revision: "$Revision: 1.11 $"

class interface
    EV_LIST_ITEM

creation
    make,
    make_with_text,
    make_with_pixmap,
    make_with_all

feature -- Access

    parent: EV_LIST
            -- Parent of the current item.
        require
            exists: not destroyed

feature -- Status report

    index: INTEGER
            -- Index of the current item.
        require
            exists: not destroyed

    is_first: BOOLEAN
```

```
            -- Is the item first in the list ?
        require
            exists: not destroyed

    is_last: BOOLEAN
            -- Is the item last in the list ?
        require
            exists: not destroyed

    is_selected: BOOLEAN
            -- Is the item selected ?
        require
            exists: not destroyed

feature -- Status setting

    set_selected (flag: BOOLEAN)
            -- Select the item if flag, unselect it otherwise.
        require
            exists: not destroyed

    toggle
            -- Change the state of selection of the item.
        require
            exists: not destroyed

feature -- Element change

    set_parent (par: EV_LIST)
            -- Make par the new parent of the widget.
            -- par can be Void then the parent is the screen.
        require
            exists: not destroyed
        ensure
            parent_set: parent = par

feature -- Implementation

    implementation: EV_LIST_ITEM_I

feature -- Event -- removing command association
```

```
        remove_double_click_commands
                -- Empty the list of commands to be executed when
                -- the item is double-clicked.
            require
                exists: not destroyed

    feature -- Event : command association

        add_double_click_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
                -- Add 'cmd' to the list of commands to be executed
                -- when the item is double clicked.
            require
                exists: not destroyed;
                valid_command: cmd /= void

    end -- class EV_LIST_ITEM
```

5.2 EV_TREE_ITEM

The parent of a tree item can be either an **EV_TREE** or an **EV_TREE_ITEM**. If the parent of the item is a tree, then it is a root item of the tree. If it is created with an item as a parent, then the item becomes a sub-tree item.

Class

```
    indexing
        description: "EiffelVision tree item. Item that can be put in a tree. A tree item is also a tree-item
    container because if we create a tree-item with a tree-item as parent, the parent will become a subtree."
        status: "See notice at end of class"
        id: "$$"
        date: "$Date: 1999/03/26 01:34:43 $"
        revision: "$Revision: 1.10 $"

    class interface
        EV_TREE_ITEM

    creation
        make,
        make_with_text,
        make_with_pixmap,
        make_with_all

    feature -- Access
```

```
        parent: EV_TREE_ITEM_HOLDER
                -- Parent of the current item.
            require
                exists: not destroyed

feature -- Status report

        is_expanded: BOOLEAN
                -- is the item expanded?
            require
                exists: not destroyed

        is_selected: BOOLEAN
                -- Is the item selected?
            require
                exists: not destroyed

feature -- Element change

        set_parent (par: EV_TREE_ITEM_HOLDER)
                -- Make par the new parent of the widget.
                -- par can be Void then the parent is the screen.
            require
                exists: not destroyed
            ensure
                parent_set: parent = par

feature -- Event -- removing command association

        remove_subtree_commands
                -- Empty the list of commands to be executed when
                -- the selection subtree is expanded or collapsed.
            require
                exists: not destroyed

feature -- Event : command association

        add_subtree_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
                -- Add cmd to the list of commands to be executed
                -- when the selection subtree is expanded or collapsed.
            require
                exists: not destroyed;
```

```
            valid_command: cmd /= void

end -- class EV_TREE_ITEM
```

5.3 EV_MENU_ITEM

The parent of a menu item can be either an **EV_MENU** or an **EV_MENU_ITEM**. The parent of a menu-item should not be a check or a radio menu item, just a simple item. If you add a menu item to another menu item, the receiver item becomes a sub-menu.

Class

```
indexing
    description: "EiffelVision menu item. Item that must be put in an EV_MENU_ITEM_HOLDER."
    status: "See notice at end of class"
    id: "$Id: ev_menu_item.e,v 1.16 1999/03/26 01:34:42 aitkaci Exp $"
    date: "$Date: 1999/03/26 01:34:42 $"
    revision: "$Revision: 1.16 $"

class interface
    EV_MENU_ITEM

creation
    make,
    make_with_text,
    make_with_pixmap,
    make_with_all

feature -- Access

    parent: EV_MENU_ITEM_HOLDER
            -- Parent of the current item.
        require
            exists: not destroyed

feature -- Status report

    insensitive: BOOLEAN
            -- Is current item insensitive to
            -- user actions?
        require
            exists: not destroyed

feature -- Status setting
```

```
        set_insensitive (flag: BOOLEAN)
                -- Set current item in insensitive mode if
                -- flag.
            require
                exists: not destroyed
            ensure
                flag = insensitive

feature -- Element change

        set_parent (par: EV_MENU_ITEM_HOLDER)
                -- Make par the new parent of the widget.
                -- par can be Void then the parent is the screen.
            require
                exists: not destroyed

feature -- Implementation

        implementation: EV_MENU_ITEM_I

end -- class EV_MENU_ITEM
```

5.4 EV_CHECK_MENU_ITEM

A check menu item is a menu item with two possible states :

- *selected*,
- *cleared*.

Class

```
indexing
        description: "EiffelVision check menu item. Item that must be put in an
EV_MENU_ITEM_HOLDER. It has two states : check and unchecked."
        status: "See notice at end of class"
        id: "$Id: ev_check_menu_item.e,v 1.6 1999/03/26 01:34:40 aitkaci Exp $"
        date: "$Date: 1999/03/26 01:34:40 $"
        revision: "$Revision: 1.6 $"

class interface
        EV_CHECK_MENU_ITEM
```

```
creation
    make,
    make_with_text

feature -- Status report

    state: BOOLEAN
            -- Is current menu-item checked ?.
        require
            exists: not destroyed

feature -- Status setting

    set_state (flag: BOOLEAN)
            -- Make flag the new state of the menu-item.
        require
            exists: not destroyed
        ensure
            correct_state: state = flag

    toggle
            -- Change the state of the menu-item to
            -- opposite
        require
            exists: not destroyed
        ensure
            state_is_true: state = not old state

feature -- Implementation

    implementation: EV_CHECK_MENU_ITEM_I

end -- class EV_CHECK_MENU_ITEM
```

5.5 EV_RADIO_MENU_ITEM

A radio item is a check item that belong to an exclusive group, only one item in the group can be selected at a time, the other items are cleared.

To have two items in the same group, you need to make one of the item the peer of the other one by using the feature *set_peer (peer: EV_RADIO_MENU_ITEM)*.

Class

indexing

description: "EiffelVision radio menu item. Item that must be put in an EV_MENU_ITEM_HOLDER. It has the same appearance than the check menu-item, yet, when a radio menu-item is checked, all the other radio menu-item of the container are unchecked."

status: "See notice at end of class"

id: "$Id: ev_radio_menu_item.e,v 1.6 1999/03/26 01:34:43 aitkaci Exp $"

date: "$Date: 1999/03/26 01:34:43 $"

revision: "$Revision: 1.6 $"

class interface

EV_RADIO_MENU_ITEM

creation

make,

make_with_text,

make_peer_with_text

feature *-- Implementation*

implementation: EV_RADIO_MENU_ITEM_I

feature *-- Status Setting*

set_peer (peer: EV_RADIO_MENU_ITEM)

-- Put in same group as peer

require

exists: ***not*** *destroyed*

ensure

implementation.is_peer (peer)

end *-- class EV_RADIO_MENU_ITEM*

5.6 EV_STATUS_BAR_ITEM

The parent of a status bar item can only be a status bar. You can set the width of this item. If you want the item to span the length of the status bar, set width to **-1**. By default the last item added has a width of **-1**.

Class

5.7 EV_MULTI_COLUMN_LIST_ROW

The parent of a multi-column list row can only be a multi-column list.

Class

indexing

description: "EiffelVision multi-column list row. These rows are used in the multi-column lists."
status: "See notice at end of class"
note: "It is not an item because it doesn%'t have the same options."
date: "$Date: 1999/03/12 20:10:12 $"
revision: "$Revision: 1.8 $"

class interface

EV_MULTI_COLUMN_LIST_ROW

creation

make,
make_with_text

feature *-- Access*

columns: INTEGER
-- Number of columns in the row
require
exists: ***not*** *destroyed*

parent: EV_MULTI_COLUMN_LIST
-- List that container this row

feature *-- Status report*

is_selected: BOOLEAN
-- Is the item selected
require
exists: ***not*** *destroyed*

feature *-- Status setting*

set_selected (flag: BOOLEAN)
-- Select the item if flag, unselect it otherwise.

```
            require
                exists: not destroyed

        toggle
                -- Change the state of selection of the item.
            require
                exists: not destroyed

feature -- Element Change

        set_cell_text (column: INTEGER; a_text: STRING)
                -- Make text  the new label of the column-th
                -- cell of the row.
            require
                exists: not destroyed;
                column_exists: column >= 1 and column <= columns;
                text_not_void: a_text /= void

        set_text (a_text: ARRAY [STRING])
            require
                exists: not destroyed;
                text_not_void: a_text /= void;
                valid_text_length: a_text.count <= columns

feature -- Event -- removing command association

        remove_activate_commands
                -- Empty the list of commands to be executed
                -- when the item is activated.
            require
                exists: not destroyed

        remove_deactivate_commands
                -- Empty the list of commands to be executed
                -- when the item is deactivated.
            require
                exists: not destroyed

feature -- Event : command association

        add_activate_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
                -- Add cmd to the list of commands to be executed
```

```
            -- when the item is activated.
        require
            exists: not destroyed;
            valid_command: cmd /= void

    add_deactivate_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed
            -- when the item is deactivated.
        require
            exists: not destroyed;
            valid_command: cmd /= void

end -- class EV_MULTI_COLUMN_LIST_ROW
```

6

Components

Some graphical components are not widgets be ause they cannot be add to a container.
In EiffelVision, we find some of this components which are described here.

6.1 EV_PIXMAP

A pixmap is a picture that contains several pixels of possibly different colors (pixmap = pixel map). It can be either read from a file or created empty and then filled. You can draw on a pixmap — drawings are then stored on the pixmap.

A pixmap can be set in all the pixmapable widgets as either buttons or items.

Class

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 1.6 1999/02/02 01:20:44 aitkaci Exp $"
date: "$Date: 1999/02/02 01:20:44 $"
revision: "$Revision: 1.6 $"

class interface
EV_PIXMAP

creation
make,
make_from_file

feature *-- Measurement*

height: INTEGER

width: INTEGER

```
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

feature -- Implementation

    implementation: EV_PIXMAP_I
            -- Implementation of pixmap

end -- class EV_PIXMAP
```

6.2 EV_SCREEN

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

6.3 EV_MENU

A menu is a rectangular area that contains a vertical list of menu items. Each menu item is of type **EV_MENU_ITEM**. The parent of a menu can be an **OPTION_BUTTON**, an **EV_STATIC_MENU_BAR** or an **EV_POPUP_MENU**.

Class

```
indexing
    description: "EiffelVision menu. Menu contains several menu items and shows them when the menu
is opened."
    status: "See notice at end of class"
    id: "$Id: ev_menu.e,v 1.11 1999/03/26 01:35:49 aitkaci Exp $"
    date: "$Date: 1999/03/26 01:35:49 $"
    revision: "$Revision: 1.11 $"

class interface
    EV_MENU

creation
    make,
```

make_with_text

feature *-- Access*

text: STRING
-- Label of the current menu
require
exists: ***not*** *destroyed*

feature *-- Element change*

set_parent (par: EV_MENU_HOLDER)
-- Make par the new parent of the item.
require
exists: ***not*** *destroyed*

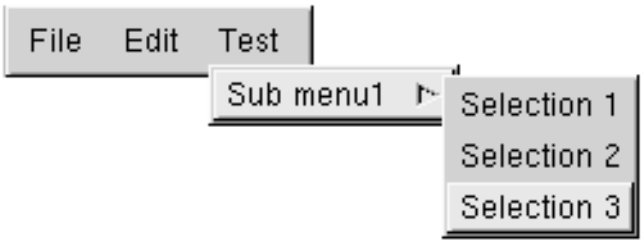
feature *-- Implementation*

implementation: EV_MENU_I

end *-- class EV_MENU*

6.4 EV_STATIC_MENU_BAR

A static menu bar displays accross the top of a window and can recive an unlimited number of menus.



indexing
description: "EiffelVision static menu bar. A menu bar that alwaysstay on the top of the window."
status: "See notice at end of class"
id: "$Id: ev_static_menu_bar.e,v 1.6 1999/03/26 01:35:49 aitkaci Exp $"
date: "$Date: 1999/03/26 01:35:49 $"
revision: "$Revision: 1.6 $"

class interface
EV_STATIC_MENU_BAR

```
creation
    make

feature -- Access

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

end -- class EV_STATIC_MENU_BAR
```

6.5 EV_STATUS_BAR

A status bar refers to the area at the bottom of a window that gives informatino. You can add items to the status bar. These items are of type **EV_STATUS_BAR_ITEM**.

Class

7

Standard Dialogs

A common dialog is a dialog that contains children which perform a specific function as opening a file or choosing a printer. All of them are modal dialogs — the application looses focus until the dialog closes. At this point, the commands execute.

7.1 EV_STANDARD_DIALOG

All dialogs inherit from the deferred class **EV_STANDARD_DIALOG**. This class contains only one feature : *show*. When you create a dialog, the options of the dialog can be changed when the window does not display.

Class

7.2 EV_MESSAGE_DIALOG

EV_MESSAGE_DIALOG is a deferred class ancestor of the four classes : **EV_QUESTION_DIALOG**, **EV_WARNING_DIALOG**, **EV_ERROR_DIALOG** and **EV_INFORMATION_DIALOG**.

These dialogs send a graphical message to the user and contain a title, a message and a combination of buttons. You can choose any of combination among the following :

- **OK**,
- **OK** and **Cancel**,
- **Yes** and **No**,
- **Yes**, **No** and **Cancel**,
- **Abort**, **Retry** and **Ignore**
- **Retry** and **Cancel**

An **Help** button can also be added to any of those combinaison.

Each dialog contains a default behavior. For example, if a dialog contains the **Cancel**, the ESC key closes the window and activates the *cancel* command. The ENTER key is equivalent to clicking the **Yes** or the **OK** button.

Class

indexing

description: "EiffelVision message dialog. Deferred class, ancestor of the standard dialogs for warning, informations, error or question."

note: "Once the dialog is create with the procedure 'make_default%' the status settings features have no effect. To use them, the dialog must be first created with 'make%' and then the user choose the buttons he wants in the dialog."

status: "See notice at end of class"

date: "$Date: 1999/03/25 20:24:51 $"

revision: "$Revision: 1.10 $"

deferred class interface

EV_MESSAGE_DIALOG

feature *-- Status report*

selected_button: STRING

-- Return the label of the selected button.

-- Can be any string in :

-- "OK", "Cancel", "Yes", "No", "Abort",

-- "Retry", "Ignore", "Help".

feature *-- Status setting*

add_help_button

-- Add an "Help" button to the other choosen buttons

-- in the dialog box.

require

exist: ***not*** *destroyed*

show_abort_retry_ignore_buttons

-- Show three buttons in the dialog: "Abort", "Retry" and "Ignore".

require

exist: ***not*** *destroyed*

show_ok_button

-- Show one button in the dialog : "OK".

require

```
            exist: not destroyed

    show_ok_cancel_buttons
            -- Show two buttons in the dilaog: "OK" and "Cancel".
        require
            exist: not destroyed

    show_retry_cancel_buttons
            -- Show two buttons in the dialog: "Retry" and "Cancel".
        require
            exist: not destroyed

    show_yes_no_buttons
            -- Show two buttons in the dialog: "Yes" and "No".
        require
            exist: not destroyed

    show_yes_no_cancel_buttons
            -- Show three buttons in the dialog: "Yes", "No" and "Cancel".
        require
            exist: not destroyed

feature -- Element change

    set_message (str: STRING)
            -- Make str the new title of the dialog.
        require
            exists: not destroyed;
            valid_message: str /= void

    set_title (str: STRING)
            -- Make str the new title of the dialog.
        require
            exists: not destroyed;
            valid_title: str /= void

feature -- Implementation

    implementation: EV_MESSAGE_DIALOG_I

feature -- Event - command association
```

```
add_abort_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed when
        -- the Abort button is pressed.
        -- If there is no Abort button, the event never occurs.
    require
        exists: not destroyed;
        valid_command: cmd /= void

add_cancel_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed when
        -- the "Cancel" button is pressed.
        -- If there is no "Cancel" button, the event never occurs.
    require
        exists: not destroyed;
        valid_command: cmd /= void

add_help_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed when
        -- the "Help" button is pressed.
        -- If there is no "Help" button, the event never occurs.
    require
        exists: not destroyed;
        valid_command: cmd /= void

add_ignore_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed when
        -- the Ignore button is pressed.
        -- If there is no Ignore button, the event never occurs.
    require
        exists: not destroyed;
        valid_command: cmd /= void

add_no_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed when
        -- the No button is pressed.
        -- If there is no No button, the event never occurs.
    require
        exists: not destroyed;
        valid_command: cmd /= void

add_ok_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
        -- Add cmd to the list of commands to be executed when
```

```
            -- the "OK" button is pressed.
            -- If there is no "OK" button, the event never occurs.
        require
            exists: not destroyed;
            valid_command: cmd /= void

    add_retry_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed when
            -- the Retry button is pressed.
            -- If there is no Retry button, the event never occurs.
        require
            exists: not destroyed;
            valid_command: cmd /= void

    add_yes_command (cmd: EV_COMMAND; arg: EV_ARGUMENT)
            -- Add cmd to the list of commands to be executed when
            -- the Yes button is pressed.
            -- If there is no Yes button, the event never occurs.
        require
            exists: not destroyed;
            valid_command: cmd /= void

feature -- Event -- removing command association

    remove_abort_commands
            -- Empty the list of commands to be executed when
            -- "Abort" button is pressed.
        require
            exists: not destroyed

    remove_cancel_commands
            -- Empty the list of commands to be executed when
            -- "Cancel" button is pressed.
        require
            exists: not destroyed

    remove_help_commands
            -- Empty the list of commands to be executed when
            -- "Help" button is pressed.
        require
            exists: not destroyed
```

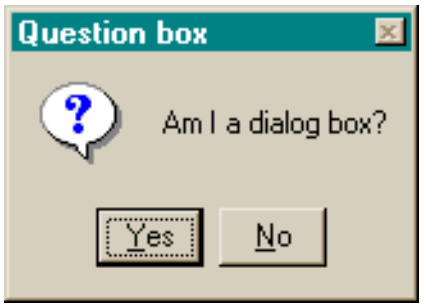
```
    remove_ignore_commands
            -- Empty the list of commands to be executed when
            -- "Ignore" button is pressed.
        require
            exists: not destroyed

    remove_no_commands
            -- Empty the list of commands to be executed when
            -- "No" button is pressed.
        require
            exists: not destroyed

    remove_ok_commands
            -- Empty the list of commands to be executed when
            -- "OK" button is pressed.
        require
            exists: not destroyed

    remove_retry_commands
            -- Empty the list of commands to be executed when
            -- "Retry" button is pressed.
        require
            exists: not destroyed

    remove_yes_commands
            -- Empty the list of commands to be executed when
            -- "Yes" button is pressed.
        require
            exists: not destroyed

end -- class EV_MESSAGE_DIALOG
```

7.3 EV_INFORMATION_DIALOG

An information dialog is a message dialog that uses the common pixmap for information. A default information dialog only contains the **OK** button.

7.4 EV_QUESTION_DIALOG

A question dialog is a message dialog that uses the common pixmap for a question (a question mark). A default question dialog contains the **Yes** and **No** buttons.



7.5 EV_WARNING_DIALOG

A warning dialog is a message dialog that uses the common pixmap for warning (a exclamation mark). A default warning dialog only contains the **OK** button.

7.6 EV_ERROR_DIALOG

An error dialog is a message dialog that uses the common pixmap for error (a red cross). A default error dialog only contains an **OK** button.

7.7 EV_FILE_SELECTION_DIALOG

EV_FILE_SELECTION_DIALOG is a deferred ancestor of both **EV_FILE_SAVE_DIALOG** and **EV_FILE_OPEN_DIALOG**.

You can use these dialogs to choose a file to save, or to retrieve some information to or from a file.

Class

7.8 EV_FILE_SAVE_DIALOG

A file save dialog is a file selection dialog that opens a file for editing.

7.9 EV_FILE_OPEN_DIALOG

A file open dialog is a file selection dialog that opens a file for reading.

7.10 EV_DIRECTORY_SELECTION_DIALOG

A directory selection dialog is a dialog that returns a directory choosen by the user.

Class

7.11 EV_FONT_SELECTION_DIALOG

A font selection dialog is a dialog used to retrieve the the choice of the user about a font and the different options of the font, as the size or the format.

Class

7.12 EV_COLOR_SELECTION_DIALOG

A color selection dialog is a dialog used to retrieve the choice of a user about a color.

Class

7.13 EV_PRINT_DIALOG

A print dialogs is a dialogs used to retrieve the choice of the user about a printer and the different options of printing.

Class