GTKlos extension

Erick Gallesio

Table of Contents

1. Introduction . . . 1

   1.1. Installation . . . 1

   1.2. Getting started . . . 2

2. Container widgets . . . 6

   2.1. Class <window> . . . 6

   2.2. Class <vwindow> . . . 6

   2.3. Class <hwindow> . . . 7

   2.4. Class <gtk-box> . . . 8

   2.5. Class <hbox> . . . 9

   2.6. Class <vbox> . . . 9

   2.7. Class <frame> . . . 9

   2.8. Class <hframe> . . . 10

   2.9. Class <vframe> . . . 10

   2.10. Class <grid> . . . 11

   2.11. Class <header-bar> . . . 12

   2.12. Class <toolbar> . . . 14

3. Display widgets . . . 17

   3.1. Class <label> . . . 17

   3.2. Class <image> . . . 17

   3.3. Class <progress-bar> . . . 18

   3.4. Class <scale> . . . 19

   3.5. Class <separator> . . . 20

4. Signal and Events . . . 21

5. Buttons . . . 22

   5.1. Class <button> . . . 22

   5.2. Class <checkbutton> . . . 22

   5.3. Class <radiobutton> . . . 22

   5.4. Class <combobox> . . . 22

   5.5. Class <menu> . . . 22

6. Entries and Text widgets . . . 23

   6.1. Class <entry> . . . 23

   6.2. Class <text> . . . 24

7. Dialog widgets . . . 26

   7.1. Class <dialog> . . . 26

   7.2. Class <color-dialog> . . . 26

   7.3. Class <file-dialog> . . . 26

   7.4. Class <font-dialog> . . . 26

8. Basic Classes . . . 27

8.1. Class <gtk-object> 27
8.2. Class <gtk-destroyed-object> 27
8.3. Class <gtk-widget> 28
8.4. Class <gtk-container> 30
8.5. Class <gtk-misc> 30
8.6. Class <gtk-orientable> 31
9. Canvases 32
Index 33

Chapter 1. Introduction

This extension permits to easily program GUI (Graphical User Interfaces) using the OO model of *STklos*. The model used here, is very similar to the one originally defined in STk Scheme and is discussed in the papers image: - Programming Graphical User Interfaces with Scheme, and in - Designing a Meta Object Protocol to wrap a Standard Graphical Toolkit.

Furthermore, this OO model allows you to define your own widgets thanks to the MOP (Meta Object Protocol) of *STklos*.

1.1. Installation

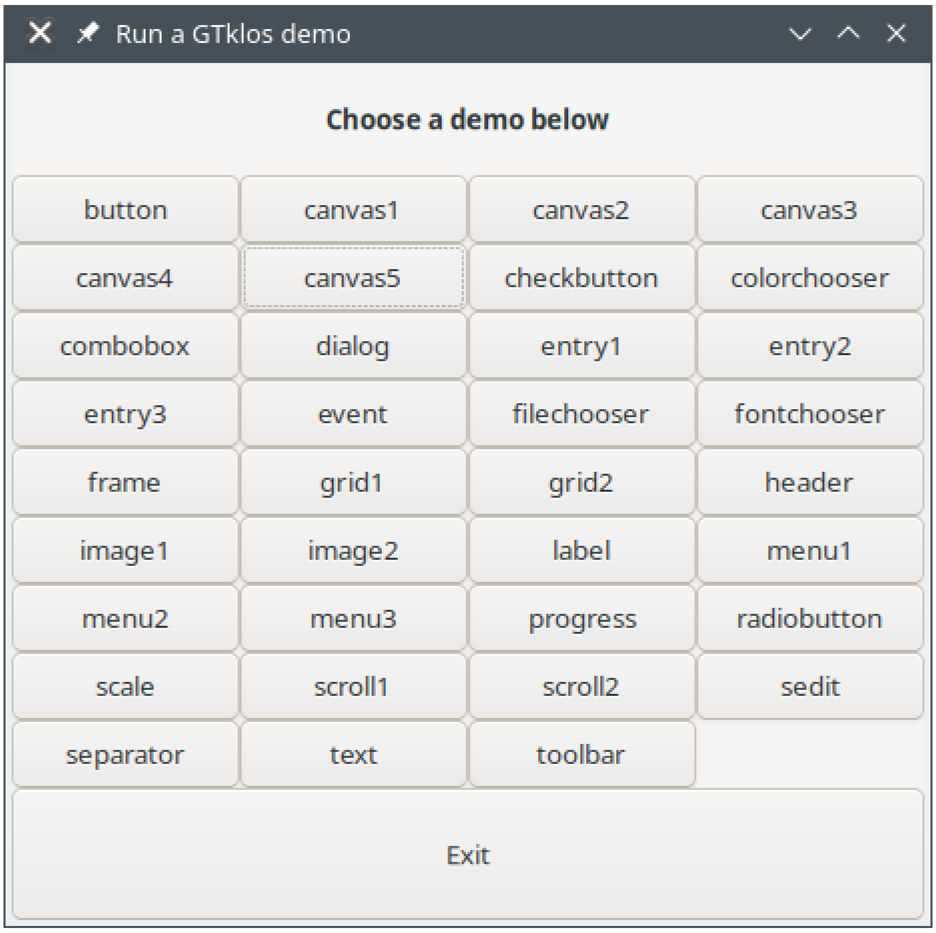
The extension is in the `extensions/gtklos` directory of *STklos*. It is configured when you run `configure` in the main directory of *STklos*. So, to compile it you just need to run

```
$ make
```

The `demos` directory contains several demos that can be run separately or with the `run-demos` script in this directory. Running this script, with

```
$ cd demos
$ ./run-demos
```

you should obtain something like:



If everything is correct, you can install the *GTklos* extension with a `make install` in the `gtklos` directory. Everything will be installed in a sub-directory of the main *STklos* installation

1.2. Getting started

To use the *GTklos* extension you need to import the `(stklos gtklos)` library. This can be done with:

```
;; Import GTKlos to access its exported symbols
(import (stklos gtklos))
```

1.2.1. A first window

The first thing you must do to make an interface consists to create an instance of the class `<window>`. For instance,

```
stklos> (define w (make <window> #:title "A first window"))
```

will create a window with a title set to `"A first window"`. You can see all the slots that can be set in `w` by using describe:

```
stklos> (describe w)
#[<window> 7470d5e89330] is an instance of class <window>.
Slots are:
     %children = ()
     %data = ()
     %event = ()
     border-width = 0
     can-default = #f
     can-focus = #f
     children = ()
     has-default = #f
     has-focus = #f
     height = 200
     height-request = -1
     modal = #f
     name = ""
     parent = #f
     resizable = #t
     sensitive = #t
     show = #t
     title = "A first window"
     tooltip = #f
     transient = #f
     visible = #t
     wid = #[gtk-window-pointer 5a473af6f1b0 @ 7470d5e89300]
     width = 200
     width-request = -1

stklos>
```

Now that the window is created, we need to start the GTK+ interaction loop to see it effectively on our screen. This can be done by calling

- `(start-interactive-gtk)`, or
- `(gtk-main)`

As said by its name, the fist form is preferred when we create an interface interactively in the REPL. This form, call the GTK+ event loop when your keyboard is idle. The second form is generally used when you create a script and dont use the REPL.

As we can see, the *width* and the *height* of this window are reflected in the `width` and `height` of `w`. Hereafter, are some manipulation with the width of `w`:

```
;; Use start-interactive-gtk to develop in the REPL
stklos> (start-interactive-gtk)
;; Setting the width to 400
stklos> (slot-set! w 'width 400)
;; Reading back the value
stklos> (slot-ref w 'width)
400
;; Since accessors are defined on all slots we can also do
stklos> (width w)
```

```
400
stklos> (set! (width w) 300)
stklos> (width w)
300
```

Of course, we can also define the widget size at creation time with a class such as

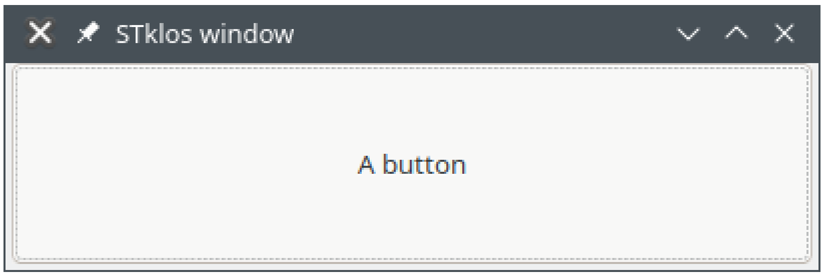
```
stklos> (define w (make <window> #:title "STklos window"
                                 #:width 400
                                 #:height 100))
```

1.2.2. Adding a button

We can add a button to a the previous window bay making an instance of a `<button>`:

```
(define b (make <button> #:parent w #:text "A button"))
```

By saying that the parent of `b` is `w`, the window we have just created just before, this button will be *inside* the `w` window. So, we will obtain:



Using describe, on `b` we have:

```
stklos> (describe b)
#[<button> 7470d5eb32a0] is an instance of class <button>.
Slots are:
     %children = ()
     %data = ()
     %event = ()
     border-width = 0
     can-default = #f
     can-focus = #t
     children = ()
     command = #f
     focus-on-click = #t
     has-default = #f
     has-focus = #f
     height = 1
     height-request = -1
     image = #f
     name = ""
     parent = #[<window> 7470d5e89330]
     relief = normal
     sensitive = #t
     show = #t
     text = "A button"
     tooltip = #f
     use-stock = #f
     use-underline = #t
     visible = #t
     wid = #[gtk-button-pointer 5a473af3f1c0 @ 7470d5eb3240]
     width = 1
     width-request = -1
     xalign = 0.5
     yalign = 0.5

stklos>
```

The slot `command` is particularly important on buttons. It contains the callback function that will be called when we click (with left mouse button) on `b`. The function will be called with two parameters the widget which has been clicked and an event object which contains all the information on the event itself (more on that below).

We can add a *command* to the previous button with:

```
stklos> (set! (command b)
              (lambda (w e)
                (printf "Button ~s was clickedn" w)))
```

Now, when clicking the button `b` a message will be printed.

This ends this small introduction on GTKlos.

Chapter 2. Container widgets

A container is a widget that can contain other widgets. It permits to create new windows or organize le layout of a multi widget GUI component.

2.1. Class <window>

The `<window>` class defines a new window. By default, a new window is mapped on the screen it is created and it’s size is 200x200.

Inherited classes:	`<gtk-container>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	`<dialog>`
	`<vwindow>`
	`<hwindow>`

Direct slots:	height modal resizable title transient width
Direct (non accessor) methods:	realize-widget (<window> <top>)

All these slots have an associated accessor that can be used to read or write the value of the slot. For instance

```
stklos> (define w (make <window> :title "A window"))
;; w
stklos> (title w)
"A window"
stklos> (set! (title w) "Title changed")
stklos> (title w)
"Title changed"
stklos>
```

2.2. Class <vwindow>

A `<vwindow>` is a utility class. It is a window which contains a vertical box, to arrange vertically some widgets in it. See example below.

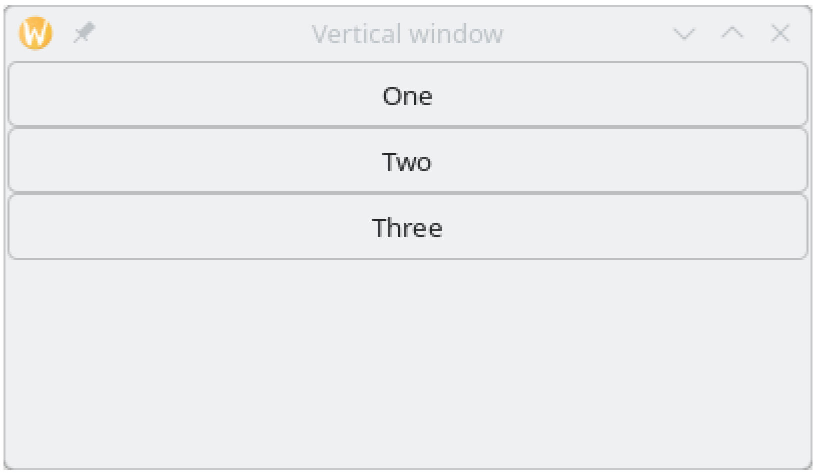
Inherited classes:	`<window>`
	`<gtk-container>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	
Direct (non accessor) methods:	

Example

```
stklos> (define w (make <vwindow> :title "Vertical window" :width 400))
;; w
stklos> (dolist (b '("One" "Two" "Three")) (make <button> :text b :parent w))
stklos>
```

This will display the following window:



Note

Specifying that a button has `w` as parent permits to embed it in the `w` container.

2.3. Class <hwindow>

A `<hwindow>` is a utility class. It is a window which contains an horizontal box, to arrange horizontally some widgets in it.

Inherited classes:	`<window>`
	`<gtk-container>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	
Direct (non accessor) methods:	

2.4. Class <gtk-box>

A `<gtk-box>` is a simple container which arranges child widgets into a single row or column, depending upon the value of `orientation` property.

⚠ Normally, a class name prefixed by `<gtk-` is an internal class which is not exported bt the GTKlos library. This class name has been choose, because the name `<box>` is already used for the normal ***STklos*** boxes (see SRFI-111).

Inherited classes:	`<gtk-orientable>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	`<vbox>`
	`<hbox>`

Direct slots:	baseline-position expand fill homogeneous padding spacing
Direct (non accessor) methods:	container-add! (<gtk-box> <gtk-widget> . <top>) realize-widget (<gtk-box> <top>)

Notes

1. One of the most important slot o this class is the slot `orientation` (not shown here, since it is inherited from the class `<gtk-orientable>`). Its value is a symbol which can be one of the symbols `horizontal` or `vertical`.
2. Method `container-add!` accepts a list of keyword parameters after the widget to add to the container. The possible values for these keyword parameters are:
   - `expand` (defaults to the value of slot `expand` slot of the box)
   - `fill` (defaults to the value of slot `fill` slot of the box)

- `padding` (defaults to the value of slot `padding` slot of the box)
- `end` (default to `#f`)

2.5. Class <hbox>

This utility class can be used to define a `<gtk-box>` whose orientation is initialized to `horizontal`.

Inherited classes:	`<gtk-box>`
	`<gtk-orientable>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	
Direct (non accessor) methods:	initialize-instance (<hbox> <top>)

2.6. Class <vbox>

This utility class can be used to define a `<gtk-box>` whose orientation is initialized to `vertical`.

Inherited classes:	`<gtk-box>`
	`<gtk-orientable>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	
Direct (non accessor) methods:	initialize-instance (<vbox> <top>)

2.7. Class <frame>

A <frame> widget surrounds its child with a decorative frame and an optional label.

Inherited classes:	`<gtk-container>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	`<vframe>`
	`<hframe>`

Direct slots:	shadow title xalign yalign
Direct (non accessor) methods:	realize-widget (<frame> <top>)

Notes

- The `title` slot contains the label of the frame
- The `shadow` slot value can be one of the following symbols `none`, `in`, `out`, `etched-in` or `etched-out`.
- The `xalign` and `yalign` flots can be used to adjust the position of the label.

2.8. Class <hframe>

This utility class permits to define a frame which contain an horizontal box that can be filled with components that are arranged horizontally.

Inherited classes:	`<frame>`
	`<gtk-container>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	
Direct (non accessor) methods:	

2.9. Class <vframe>

This utility class permits to define a frame which contain an vframe box that can be filled with components that are arranged vertically.

Inherited classes:	`<frame>`
	`<gtk-container>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	
Direct (non accessor) methods:	

2.10. Class <grid>

A `<grid>` widget is a container which arranges its child widgets in rows and columns, with arbitrary positions and horizontal/vertical spans.

Inherited classes:	`<gtk-container>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	column-homogeneous column-spacing row-homogeneous row-spacing
Direct (non accessor) methods:	container-add! (<grid> <gtk-widget> . <top>) realize-widget (<grid> <top>)

Note

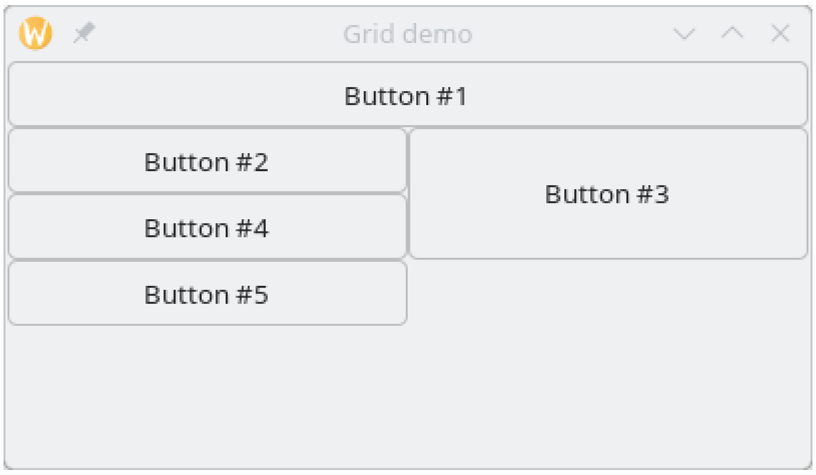
Method `container-add!` accepts a list of keyword parameters after the widget to add to the container. The possible value for these keyword parameters are:

- `left` is the column position of the added widget (starting from 1)
- `top` is the line position of the added widget (starting from 1)
- `width` is the width position of the added widget
- `height` is the height position of the added widget

Example

```
stklos> (define w (make <window> :title "Grid demo"))
;; w
stklos> (define g (make <grid> :parent w))
;; g
;; Create 5 buttons
stklos> (define b (map (lambda (x) (make <button> :text x :width 200))
                  '("Button #1" "Button #2" "Button #3" "Button #4" "Button #5")))
;; b
;; Add them to the grid
stklos> (container-add! g (list-ref b 0) #:left 0 #:top 0 :width 2)
stklos> (container-add! g (list-ref b 1) #:left 0 #:top 1)
stklos> (container-add! g (list-ref b 2) #:left 1 #:top 1 :height 2)
stklos> (container-add! g (list-ref b 3) #:left 0 #:top 2)
stklos> (container-add! g (list-ref b 4) #:left 0 #:top 3)
stklos>
```

This will display the following window:



2.11. Class <header-bar>

A `<header-bar>` is similar to a horizontal `<gtk-box>`. Furthermore, this widget can add typical window frame controls, such as minimize, maximize and close buttons, or the window icon. It is often used at the top of a `<vwindow>`

Inherited classes:	`<gtk-container>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	decoration-layout decoration-layout-set has-subtitle show-close-button subtitle title
Direct (non accessor) methods:	realize-widget (<header-bar> <top>)

Notable slots

- **decoration-layout** is a string used to indicate the layout of the buttons (see example below)
- **decoration-layout-set** is a boolean used to know if the decoration layout has been set
- **show-close-button** indicates if the decoration buttons (not only the close button!!) are shown or not. Its default value is `#f`

Exemple

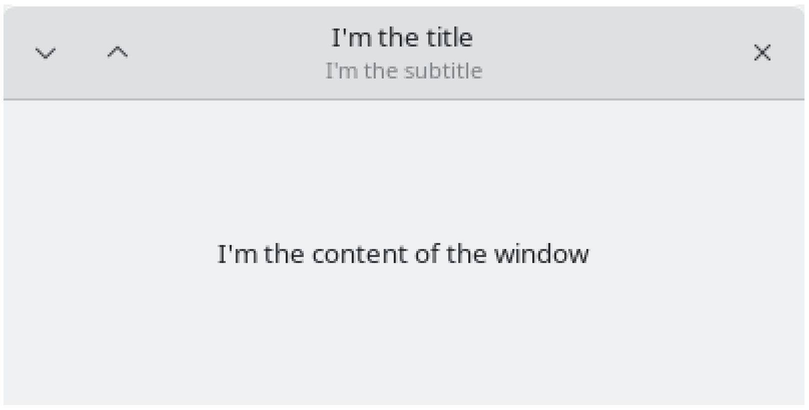
The following example illustrates the use of a header bar.

```
(define w (make <vwindow> :width 400))

(define h (make <header-bar> :title "I'm the title"
                             :subtitle "I'm the subtitle"
                             :parent w
                             :decoration-layout "minimize,maximize:close"
                             :show-close-button #t))

(define l (make <label> :text "I'm the content of the window"
                        :parent (list w :expand #t)))
```

Execution of this code will display the following window



Notes

1. The `decoration-layout` slot is set here to `"minimize, maximize:close"` to place
   - the *minimize* and *maximize* buttons on the left (since they are before the `':'` character)
   - the *close* button on the right (since it is after the `':'` character)
2. The `show-close-button` is set to `#t` so display the control buttons
3. The `parent` is set here to `w` with an indication that it must be expanded into it container (`w` here). See the documentation of `parent` of the `<gtk-widget>` class.

2.12. Class <toolbar>

A `<toolbar>` is container whose constituents are instance of the `<toolbar-item>` class.

Inherited classes:	`<gtk-container>`
	`<gtk-orientable>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	expand icon-size show-arrow toolbar-style
Direct (non accessor) methods:	add-items-to-toolbar (<toolbar> <top>) container-add! (<toolbar> <toolbar-item> <integer>) container-add! (<toolbar> <toolbar-item>) realize-widget (<toolbar> <top>)

Slots

- **expand** is a boolean. It indicates if toolbar items are expanded or not (default to `#f`)
- **icon-size** can be one of the following symbols `small`, `medium`, `large` or `huge`.
- **show-arrow** is the boolean. It indicates if the toolbar as an overflow menu.
- **toolbar-style**: can be one of the following symbols `icons`, `text`, `both` or `both-horizontal`.

Notable method

The method **add-items-to-toolbar** is a utility method to easily populate the components of a toolbar. It takes a toolbar and a list describing its components with the following convention, for each item of the list:

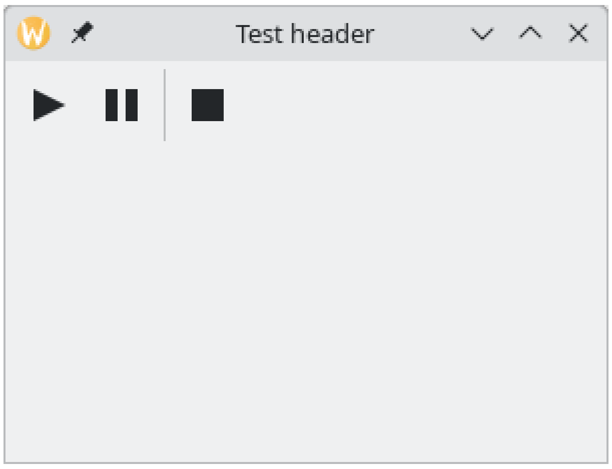
1. an empty list specify to add a new separator, that is an instance of `<toolbar-separator-item>`, to the toolbar
2. a list specifies that a new `<toolbar-icon-item>` must be created and added to the toolbar. The content of the list are the parameters that must be passed during the creation of the icon.

```
(define w (make <vwindow> :title "Test header" :width 300))
(define tb (make <toolbar> :parent w))

(add-items-to-toolbar tb                           ;; populate the toolbar
 '((:text "Play"  :icon-name "media-playback-start")
   (:text "Pause" :icon-name "media-playback-pause")
   ()  ;; <== A separator
```

```
    (:text "Stop" :icon-name "media-playback-stop")))
```

Execution of this code will display the following window



Notes

1. For the sake of simplicity, the buttons are inactive here (use the `command` slot to add an action when the toolbar button is clicked).
2. Icons her are stock buttons they are searched by the GTK library in the standard directory (generally `/usr/share/icons` on GNU/Linux).
3. Since `<gtk-toolbar>` inherits from `<gtk-orientable>`, a toobar can be horizontal or vertical.

2.12.1. Class <toolbar-item>

The `<toolbar-item>` class is the parent class of the toolbar items classes that can be added to a to a GTK toolbar. It offers only one method (**container-add**) to add an item to a toolbar.

Inherited classes:	`<gtk-widget>` `<gtk-object>`
Directly inheriting classes:	`<toolbar-button-item>` `<toolbar-separator-item>`
Direct slots:	
Direct (non accessor) methods:	container-add! (<toolbar> <toolbar-item> <integer>) container-add! (<toolbar> <toolbar-item>)

There are two methods of the generic function **container-add!** to add an item to a container:

- with 2 parameters, the methods permit to append the new item at the end of already added items.
- with 3 parameters, the method adds the given item at the position given as third parameter (an integer). If the position is 0 the item is prepended to the start of the toolbar. If it is negative, the

item is appended to the end of the toolbar.

2.12.2. Class <toolbar-separator-item>

The class `<toolbar-separator-item>` permits to define a separator to a toolbar.

Inherited classes:	`<toolbar-item>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	
Direct (non accessor) methods:	realize-widget (<toolbar-separator-item> <top>)

2.12.3. Class <toolbar-button-item>

The class `<toolbar-separator-item>` permits to define a button to a toolbar. The button can have an image and a text.

Inherited classes:	`<toolbar-item>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	command icon-name text
Direct (non accessor) methods:	realize-widget (<toolbar-button-item> <top>)

Methods

- **command** is identical to the command associated to a button. See the documentation of buttons.
- **icon-name** is a string which contains is the name of the themed icon displayed on the item. Icons are searched by the GTK library in the standard directory (generally `/usr/share/icons` on GNU/Linux).
- **text** is the text of the button item.

Chapter 3. Display widgets

A display widget is a GUI component which shows some information (a text, an image, or a progress bar for instance)

3.1. Class <label>

A `<label>` permits to define a text widget which displays a small text.

Inherited classes:	`<gtk-misc>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	justify lines selectable text
Direct (non accessor) methods:	realize-widget (<label> <top>)

Notes

- The `text` slot contains the text of the label.
- The `justify` slot may contain one of the following symbols: `left`, `right`, `center` or `fill`

3.2. Class <image>

The class `<image>` permits to display an image.

Inherited classes:	`<gtk-misc>`
	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	file-name icon-name icon-size
Direct (non accessor) methods:	get-image-pixbuf (<image>) realize-widget (<image> <top>)

Slots

- **file-name** designates a file containing the image to display. If the file cannot be found. a "broken image" icon will be displayed. If the file contains an animation, the image will be animated. The `image-path` parameter object is used to find the image file (see below).
- **icon-name**: is a string which contains is the name of the themed icon displayed on the item. Icons are searched by the GTK library in the standard directory (generally `/usr/share/icons` on GNU/Linux).
- **icon-size** can be one of the following symbols `small`, `medium`, `large` or `huge`.

Parameter object

- **image-path** is a parameter object which denotes the paths to be searched when specifying a file name with the `file-name` slot. The default path contains the current directory and a directory depending of ***STklos*** installation directory. Alternatively, the default path can also be specified with the shell variable `STKLOS_INDEX_TERM`.

Notable Method

- **get-image-pixbuf** return a C pointer on the pixbuf used to represent the image. This can be useful to insert an image in a canvas.

3.3. Class <progress-bar>

A `<progress-bar>` ca be used to display the progress of a long running operation; it can be used in two different modes: *percentage* mode and *activity mode*.

Inherited classes:	`<gtk-widget>`
	`<gtk-orientable>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	inverted pulse-step show-text text value
Direct (non accessor) methods:	progress-bar-pulse (<progress-bar>) realize-widget (<progress-bar> <top>)

Slots

- **inverted** is a boolean. It permits to invert the growing direction of a progress bar (form top to bottom for vertical progress bars and from left to right for horitontal ones).
- **pulse-step** indicates the fraction of the progress bar used to advance to the next step.
- **show-text** is a boolean used to indicate if a text is shown in the progress bar. The shown text is either the value of the `text` slot or, if not set, the value of the `value` slot, as a percentage.

- **text** is a string. It is the legend of the progress bar.
- **value** denotes the fraction of the task which is already done. Setting this slot permit to use the progress bar in *percentage* mode.

Notable method

- **progress-bar-pulse** can be used to indicate that some progress has been made. The progress bar enters “activity mode,” where a block bounces back and forth.

3.4. Class <scale>

The class `<scale>` defines a slider control used to select a numeric value.

Inherited classes:	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	command digits draw-value has-origin increment lower orientation upper value value-pos
Direct (non accessor) methods:	realize-widget (<scale> <top>)

Slots

- **command** contains a function that will be called when the value of the scale is changed. See general description on the the *command* slot in Chapter 4.
- **digits** specifies the number of decimal places that are displayed in the value.
- **draw-value** is a boolean. It indicates whether the current value is displayed.
- **has-origin** is a boolean. It indicates if the scale has an origin.
- **increment**
- **lower** is the minimum possible value of the scale.
- **orientation** indicates the orientation of the scale. Warning: it is a **read-only** slot.
- **upper** is the maximum possible value of the scale.
- **value** is the value of the scale. Setting it will move the scale.
- **value-pos** indicates the position where the value is displayed. Its value can be the symbol

top, bottom, left or right.

3.5. Class <separator>

A <separator> is a horizontal or vertical separator widget, used to group the widgets within a window. It displays a line with a shadow to make it appear sunken into the interface.

Inherited classes:	<gtk-orientable> <gtk-widget> <gtk-object>
Directly inheriting classes:	
Direct slots:	
Direct (non accessor) methods:	realize-widget (<separator> <top>)

Chapter 4. Signal and Events

TODO

Chapter 5. Buttons

TODO

5.1. Class <button>

TODO

5.2. Class <checkbutton>

TODO

5.3. Class <radiobutton>

TODO

5.4. Class <combobox>

TODO

5.5. Class <menu>

TODO

Chapter 6. Entries and Text widgets

6.1. Class <entry>

An `<entry>` widget is a single line text entry widget.

Inherited classes:	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	activates-default caps-lock-warning cursor-position has-frame max-length placeholder-text primary-icon-name secondary-icon-name text-editable text-overwrite text-visibility value
Direct (non accessor) methods:	realize-widget (<entry> <top>)

Slots

- **activates-default** is a boolean which tells if the entry will activate the default widget.
- **caps-lock-warning** is a boolean which tells if a password entry will show a warning when Caps Lock is on.
- **cursor-position** indicates the position of the cursor in the entry.
- **has-frame** is a boolean which tells if the entry has a frame.
- **max-length** is the maximum length of the entry.
- **placeholder-text** is the string which is displayed in the entry when it is empty and unfocused.
- **primary-icon-name** is the name of the icon to use for the primary icon for the entry. Icons are searched by the GTK library in the standard directory (generally `/usr/share/icons` on GNU/Linux).
- **secondary-icon-name** is the name of the icon to use for the secondary icon for the entry.
- **text-editable** is a boolean which indicates if the text entry is modifiable.
- **text-overwrite** is a boolean which tells if the text is overwritten when typing in the entry.

- **text-visibility** is a boolean which tells if the text in the entry is visible. Setting the visibility to false is useful to read secrets.
- **value** is a string which contains the text in the entry.

6.2. Class <text>

The `<text>` widget permits to create an editable text widget.

Inherited classes:	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	
Direct slots:	accepts-tab cursor-visible justify text-buffer text-editable text-indent text-monospace text-overwrite text-wrap value
Direct (non accessor) methods:	event-connect (<text> <top> <top>) event-connect-after (<text> <top> <top>) realize-widget (<text> <top>) text-copy-clipboard (<text>) text-cut-clipboard (<text>) text-paste-clipboard (<text>)

Slots

- **accepts-tab** indicates if the text widget insert a tab character when the `Tab` key is pressed.
- **cursor-visible** indicates if the cursor is displayed.
- **justify** is one of the following symbols: `left`, `right`, `center` or `fill`.
- **text-buffer** contains a C pointer to the representation of the internal GTK buffer (handle with care).
- **text-editable** is a boolean which indicates if the text widget is modifiable.
- **text-indent** contains the value of the default indentation for paragraphs.
- **text-monospace** is a boolean which indicates that the text content should use monospace fonts.
- **text-overwrite** is a boolean which tells if the text is overwritten when typing in the text

widget.

- **text-wrap** is a boolean. It indicates if text must be wrapped in the widget
- **value** contains the characters of the text widget

Methods

- **event-connect** permits to connect a signal to the text widget (see the Chapter 4)
- **text-copy-clipboard** copies the content of text clipboard
- **text-cut-clipboard** cuts the selected clipboard
- **text-paste-clipboard** pastes the clipboard in the text widget

Chapter 7. Dialog widgets

7.1. Class <dialog>

7.2. Class <color-dialog>

7.3. Class <file-dialog>

7.4. Class <font-dialog>

Chapter 8. Basic Classes

This section describes the basic classes which are inherited by the high level GTKlos widgets. These classes are not exported by the GTKlos library. However, since the slots (an their accessor function) are available in the library widgets, they are exposed here. Furthermore, all the methods described here are also available in user programs, once the library has been imported.

8.1. Class <gtk-object>

The `<gtk-object>` class is the root of the hierarchy of all the GTK object.

Inherited classes:	
Directly inheriting classes:	`<gtk-widget>`
Direct slots:	
Direct (non accessor) methods:	event-connect (<gtk-object> <top> <top>) event-connect-after (<gtk-object> <top> <top>)

Note

The `event-connect` and `event-connect-after` methods are described in the Events section.

8.2. Class <gtk-destroyed-object>

The `<gtk-destroyed-object>` class is the class given to a `<gtk-object>` which has been destroyed with the **destroy** method.

Inherited classes:	
Directly inheriting classes:	
Direct slots:	
Direct (non accessor) methods:	

8.3. Class <gtk-widget>

The `<gtk-widget>` is the base class all widgets in GTK derive from.

Inherited classes:	`<gtk-object>`

Directly inheriting classes:	`<text>`
	`<entry>`
	`<gtk-adjustement>`
	`<scale>`
	`<progress-bar>`
	`<toolbar-item>`
	`<gtk-container>`
	`<gtk-orientable>`
	`<gtk-misc>`

Direct slots:	can-default can-focus has-default has-focus height height-request name parent sensitive show tooltip visible wid width width-request
Direct (non accessor) methods:	container-add! (<grid> <gtk-widget> . <top>) container-add! (<gtk-container> <gtk-widget> . <top>) container-add! (<dialog> <gtk-widget> . <top>) container-add! (<gtk-box> <gtk-widget> . <top>) container-info (<gtk-widget>) container-remove! (<gtk-container> <gtk-widget>) destroy (<gtk-widget>) initialize-instance (<gtk-widget> <top>) internal-arrange-widget (<gtk-widget> <top>) realize-widget (<gtk-widget> <top>)

Notable slots

- **name** denotes the name of the widget. The name of the widget allows you to refer to it from a CSS file. See GTK documentation for more information.

- **parent** denotes the parent of the container widget which contain this window. A list can be used when setting the parent of a widget. In this case, the first element of the list must be the container in which the widget must be added, the rest are the parameters that would be used when using the **container-add!** method (see below).
- **sensitive** indicates the if the user can interact with it. If the widget is non sensitive, it is grayed out.
- **show** is a read only slots. It indicates if the widget is shown when realized. The default value of this slot is `#t`.
- **tooltip** is a string that can the text to be used as a tooltip for the

created widget. - **visible** is a boolean to set/unset the visibility of the widget. - **wid** is a ***STklos*** slot. It contains the low level GTK widget which implements the high level GTKlos object. Its value is generally set in the **realize-widget** method. Normal user program shouldn't change the value of this slot.

Methods

- **container-add!** is the method used to add a widget to a container. Its first argument is the container widget and its second argument is the widget to add to the container. Subsequent parameters depend of the container (each container has its own conventions to add a component to it).
- **container-info** returns some information on the way the widget has been added to its container as a list. If the widget has no parent, it returns `#f`.
- **container-remove!** permit to remove the widget form it container. The widget is not destroyed.
- **destroy** permits to destroy the widget (and all its children if it is a container). When a widget is destroyed, its class is changed to `<destroyed-gtk-object>`.
- **internal-arrange-widget** is a hook called when the widget is initialized. Most of the time it does nothing.
- **realize-widget** is the method called to create a low level GTK widget, and initialize it properly. Each widget has it own `realize-widget`. For `<gtk-widget>`, it does nothing.

8.4. Class <gtk-container>

The `<gtk-container>` class is inherited by all the container widgets.

Inherited classes:	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	`<gtk-menu-item>`
	`<gtk-menu-shell>`
	`<combobox>`
	`<button>`
	`<scroll>`
	`<toolbar>`
	`<header-bar>`
	`<grid>`
	`<frame>`
	`<window>`

Direct slots:	border-width children
Direct (non accessor) methods:	container-add! (<gtk-container> <gtk-widget> . <top>) container-remove! (<gtk-container> <gtk-widget>) destroy (<gtk-container>)

The direct methods of this class are described in the section about **<gtk-widget>** class.

8.5. Class <gtk-misc>

This class factorizes properties which are common between the label and image widgets.

Inherited classes:	`<gtk-widget>`
	`<gtk-object>`

Directly inheriting classes:	`<label>`
	`<image>`

Direct slots:	xalign xpad yalign ypad
Direct (non accessor) methods:	

8.6. Class <gtk-orientable>

The class `<gtk-orientable>`is inherited by classes which can be horizontally or vertically oriented.

Inherited classes:	`<gtk-widget>` `<gtk-object>`
Directly inheriting classes:	`<separator>` `<progress-bar>` `<toolbar>` `<gtk-box>`
Direct slots:	orientation
Direct (non accessor) methods:	

Note

The only slot of this class (**orientation**) indicates the orientation of the widget. It's value is a symbol whose value can be `horitontal` or `vertical`.

Chapter 9. Canvases

TODO

Index

A

accepts-tab *slot*, 24
activates-default *slot*, 23
add-items-to-toolbar *method*, 14

B

baseline-position *slot*, 8
border-width *slot*, 30

C

can-default *slot*, 28
can-focus *slot*, 28
caps-lock-warning *slot*, 23
children *slot*, 30
column-homogeneous *slot*, 11
column-spacing *slot*, 11
command *slot*, 16, 19
container-add! *method*, 8, 11, 14, 14, 15, 15, 28, 28, 28, 28, 30
container-info *method*, 28
container-remove! *method*, 28, 30
cursor-position *slot*, 23
cursor-visible *slot*, 24

D

decoration-layout *slot*, 12
decoration-layout-set *slot*, 12
destroy *method*, 28, 30
digits *slot*, 19
draw-value *slot*, 19

E

entry
  `<entry>` *class*, 23
event-connect *method*, 24, 27
event-connect-after *method*, 24, 27
expand *slot*, 8, 14

F

file-name *slot*, 17
fill *slot*, 8
frame
  `<frame>` *class*, 9

G

get-image-pixbuf *method*, 17
grid
  `<grid>` *class*, 11
gtk-box
  `<gtk-box>` *class*, 8
gtk-container
  `<gtk-container>` *class*, 30
gtk-destroyed-object
  `<gtk-destroyed-object>` *class*, 27
gtk-misc
  `<gtk-misc>` *class*, 30
gtk-object
  `<gtk-object>` *class*, 27
gtk-orientable
  `<gtk-orientable>` *class*, 31
gtk-widget
  `<gtk-widget>` *class*, 28

H

has-default *slot*, 28
has-focus *slot*, 28
has-frame *slot*, 23
has-origin *slot*, 19
has-subtitle *slot*, 12
hbox
  `<hbox>` *class*, 9
header-bar
  `<header-bar>` *class*, 12
height *slot*, 6, 28
height-request *slot*, 28
hframe
  `<hframe>` *class*, 10
homogeneous *slot*, 8
hwindow
  `<hwindow>` *class*, 7

I

icon-name *slot*, 16, 17
icon-size *slot*, 14, 17
image
  `<image>` *class*, 17
image-path *parameter object*, 18
increment *slot*, 19
initialize-instance *method*, 9, 9, 28

internal-arrange-widget *method*, 28
inverted *slot*, 18

J

justify *slot*, 17, 24

L

label
    `<label>` *class*, 17
lines *slot*, 17
lower *slot*, 19

M

max-length *slot*, 23
modal *slot*, 6

N

name *slot*, 28

O

orientation *slot*, 19, 31

P

padding *slot*, 8
parent *slot*, 28
placeholder-text *slot*, 23
primary-icon-name *slot*, 23
progress-bar
    `<progress-bar>` *class*, 18
progress-bar-pulse *method*, 18
pulse-step *slot*, 18

R

realize-widget *method*, 6, 8, 9, 11, 12, 14, 16, 16, 17, 17, 18, 19, 20, 23, 24, 28
resizable *slot*, 6
row-homogeneous *slot*, 11
row-spacing *slot*, 11

S

scale
    `<scale>` *class*, 19
secondary-icon-name *slot*, 23
selectable *slot*, 17
sensitive *slot*, 28
separator
    `<separator>` *class*, 20
shadow *slot*, 9
show *slot*, 28
show-arrow *slot*, 14
show-close-button *slot*, 12
show-text *slot*, 18
spacing *slot*, 8
STKLOS_IMAGE_PATH
    `STKLOS_IMAGE_PATH` shellvariable, 18
subtitle *slot*, 12

T

text
    `<text>` *class*, 24
text *slot*, 16, 17, 18
text-buffer *slot*, 24
text-copy-clipboard *method*, 24
text-cut-clipboard *method*, 24
text-editable *slot*, 23, 24
text-indent *slot*, 24
text-monospace *slot*, 24
text-overwrite *slot*, 23, 24
text-paste-clipboard *method*, 24
text-visibility *slot*, 23
text-wrap *slot*, 24
title *slot*, 6, 9, 12
toolbar
    `<toolbar>` *class*, 14
toolbar-button-item
    `<toolbar-button-item>` *class*, 16
toolbar-item
    `<toolbar-item>` *class*, 15
toolbar-separator-item
    `<toolbar-separator-item>` *class*, 16
toolbar-style *slot*, 14
tooltip *slot*, 28
transient *slot*, 6

U

upper *slot*, 19

V

value *slot*, 18, 19, 23, 24
value-pos *slot*, 19
vbox
    `<vbox>` *class*, 9
vframe
    `<vframe>` *class*, 10
visible *slot*, 28
vwindow

`<vwindow>` *class*, 6

W

wid *slot*, 28
width *slot*, 6, 28
width-request *slot*, 28
window
`<window>` *class*, 6

X

xalign *slot*, 9, 30
xpad *slot*, 30

Y

yalign *slot*, 9, 30
ypad *slot*, 30