Garnet Gadgets
Reference Manual

Andrew Mickish
Brad A. Myers
Rajan Parthasarathy

December 1994

Abstract

The Garnet Gadget Set contains common user interface objects which can be customized for use in an interface. Because the objects are extremely versatile, they may be employed in a wide range of applications with a minimum of modification. Examples of provided gadgets include menus, buttons, scroll bars, sliders, and gauges.

This research was sponsored by NCCOSC under Contract No. N66001-94-C-6037, Arpa Order No. B326. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NCCOSC or the U.S. Government.

1. Introduction

Many user interfaces that span a wide variety of applications usually have several elements in common. Menus and scroll bars, for example, are used so frequently that an interface designer would waste considerable time and effort recreating those objects each time they were required in an application.

The intent of the Garnet Gadget Set is to supply several frequently used objects that can be easily customized by the designer. By importing these pre-constructed objects into a larger Garnet interface, the designer is able to specify in detail the desired appearance and behavior of the interface, while avoiding the programming that this specification would otherwise entail.

This document is a guide to using the Gadget Set. The objects were constructed using the complete Garnet system, and their descriptions assume that the reader has some knowledge of KR, Opal, Interactors, and Aggregadgets.

1.1. Current Gadgets

Most of the gadgets described in this manual are pictured in figures 1-1 through 1-4

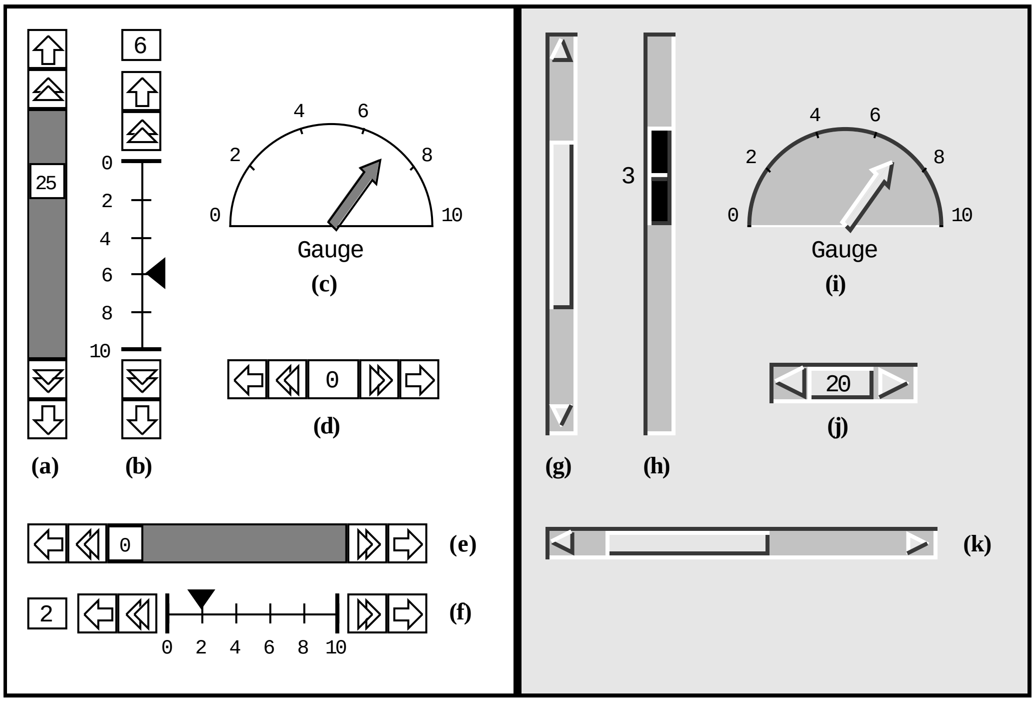


Figure 1-1: The Garnet-style and Motif-style scroll bars, sliders, and gauges. **(a)** `v-scroll-bar`, **(b)** `v-slider`, **(c)** `gauge`, **(d)** `trill-device`, **(e)** `h-scroll-bar`, **(f)** `h-slider`, **(g)** `motif-v-scroll-bar`, **(h)** `motif-slider`, **(i)** `motif-gauge`, **(j)** `motif-trill-device`, **(k)** `motif-slider`

- **Gadgets used to choose a value from a range of values**

  `v-scroll-bar` - Vertical scroll bar (p. 361)

  `v-slider` - Vertical slider (same idea as a scroll bar, but with a tic-marked shaft rather than a rectangular bounding box) (p. 363)

  `gauge` - Semi-circular gauge (the needle on the gauge may be moved to select a value) (p. 366) `trill-device` - Number input box with increment/decrement trill boxes (p. 365)

`h-scroll-bar` - Horizontal scroll bar (p. 361)

`h-slider` - Horizontal slider (p. 363)

`motif-v-scroll-bar` - Vertical scroll bar (p. 428)

`motif-slider` - Vertical slider (same idea as a scroll bar, but with text beside the indicator showing the current value) (p. 430)

`motif-gauge` - Semi-circular gauge (p. 432)

`motif-trill-device` - Number input with trill boxes (p. 431)

`motif-h-scroll-bar` - Horizontal scroll bar (p. 428)

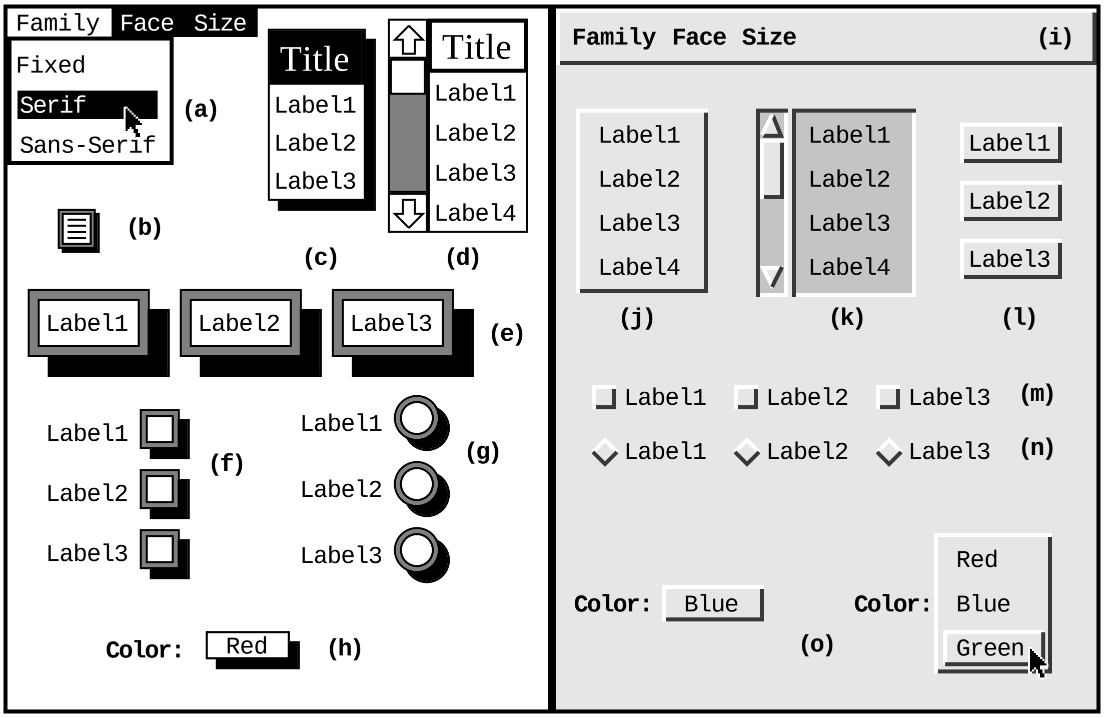


Figure 1-2: The Garnet-style and Motif-style buttons and menus. **(a)** `menubar`, **(b)** `popup-menu-button`, **(c)** `menu`, **(d)** `scrolling-menu`, **(e)** `text-button-panel`, **(f)** `x-button-panel`, **(g)** `radio-button-panel`, **(h)** `option-button`, **(i)** `motif-menubar`, **(j)** `motif-menu`, **(k)** `motif-scrolling-menu`, **(l)** `motif-text-button-panel`, **(m)** `motif-check-button-panel`, **(n)** `motif-radio-button-panel`, **(o)** `motif-option-button` in its unselected and selected state

- **Gadgets used to choose items from a list of possible choices**

`menubar` - A pull-down menu (p. 378)

`popup-menu-button` - A button which pops up a menu when pressed. The appearance of the button does not change with the selection. (p. 373)

`menu` - Vertical menu, single selection (p. 374)

`scrolling-menu` - A menu with a scroll bar on one side, which allows a subset of all items in the menu to be viewed. (single or multiple selection) (p. 376)

`text-buttons` - A panel of rectangular buttons, each with a choice centered inside the button. As an option, the currently selected choice may appear in inverse video. (single selection) (p. 367 and 369)

`x-buttons` - A panel of square buttons, each with a choice beside the button. An "X" appears inside each currently selected button. (multiple selection) (p. 367 and 370)

`radio-buttons` - A panel of circular buttons, each with a choice beside the button. A black circle appears inside the currently selected button. (single selection) (p. 367 and 371)

`option-button` - A button which pops up a menu when pressed. Selection of a choice from the menu causes that item to appear as the new label of the button. (p. 371)

`motif-menubar` - A pull-down menu. (p. 441)

`motif-menu` - Vertical menu, single selection (p. 438)

`motif-scrolling-menu` - A menu with an attached scroll bar. (p. 440)

`motif-text-buttons` - A panel of rectangular buttons, each with a choice appearing inside the button. (single selection) (p. 433 and 434)

`motif-check-buttons` - A panel of square buttons, each with a choice beside the buttons. (multiple selection) (p. 433 and 435)

`motif-radio-buttons` - A panel of diamond buttons, each with a choice beside the button. (single selection) (p. 433 and 436)

`motif-option-button` - A button which pops up a menu when pressed. Selection of a choice from the menu causes that item to appear as the new label of the button. (p. 437)

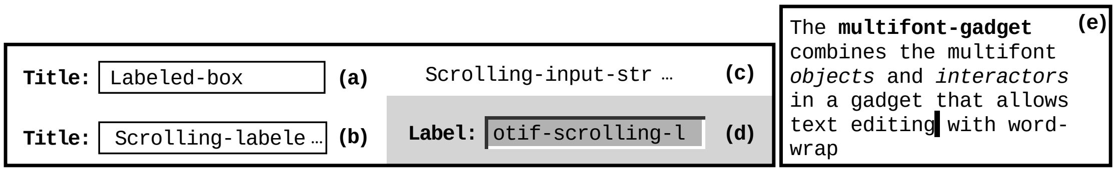


Figure 1-3: Text gadgets. **(a)** `labeled-box`, **(b)** `scrolling-labeled-box`, **(c)** `scrolling-input-string`, **(d)** `motif-scrolling-labeled-box`, **(e)** `multifont-gadget`

- **Gadgets used to handle text input**

`labeled-box` - A framed text object that may be edited. As the string gets longer, the frame expands. (p. 384)

`scrolling-labeled-box` - A scrolling input string in a box with a label. The frame stays fixed, and the string scrolls. (p. 385)

`scrolling-input-string` - Input a text string, but using a fixed width area and scroll the string horizontally if necessary. (p. 385)

`motif-scrolling-labeled-box` - A labeled box with text inside that may be edited. (p. 448)

`multifont-gadget` - A text editing gadget that includes word wrap, text selection, and many functions that allow manipulation of the text. This gadget is discussed in the Opal manual.

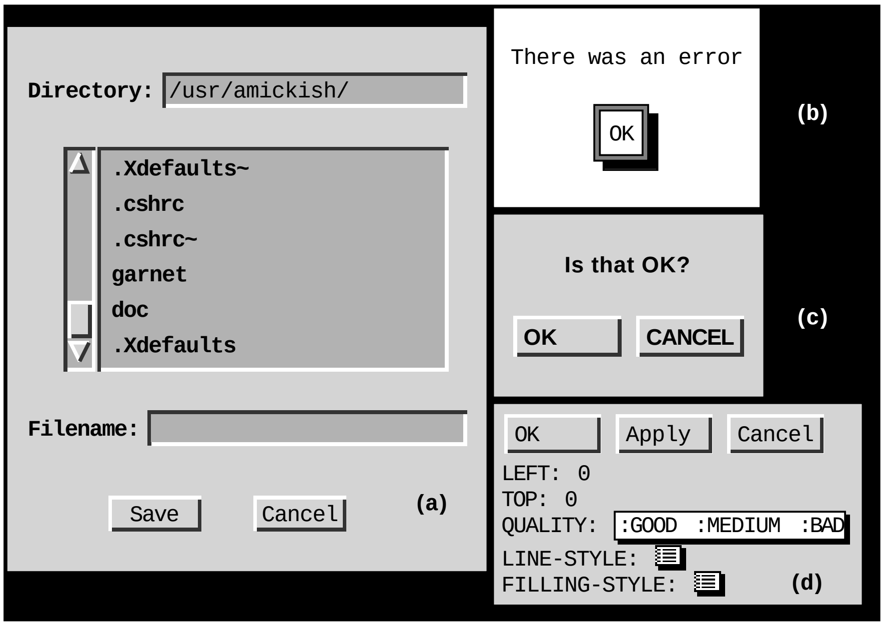


Figure 1-4: Garnet dialog boxes. **(a)** `motif-save-gadget`, **(b)** `error-gadget`, **(c)** `motif-query-gadget`, **(d)** `motif-prop-sheet-with-OK`

- **Dialog boxes for reading and writing to files** (the `motif-save-gadget` is pictured in figure 1-4)

  `save-gadget` - Saves a file in a directory whose contents are displayed in a scrolling menu. (p. 407)

  `load-gadget` - Loads a file from a directory whose contents are displayed in a scrolling menu. (p. 411)

  `motif-save-gadget` - Saves a file in a directory whose contents are displayed in a Motif style scrolling menu. (p. 450)

  `motif-load-gadget` - Loads a file from a directory whose contents are displayed in a Motif style scrolling menu. (p. 451)

- **Dialog boxes for reporting errors to the user and asking for user input** (the `error-gadget` and `motif-query-gadget` are pictured in figure 1-4).

  `error-gadget` - Used to display error messages in a window with an "OK" button (p. 403)

  `query-gadget` - A dialog box like the `error-gadget`, but with multiple buttons and the ability to return values. (p. 406)

  `motif-error-gadget` - A dialog box used to display error messages with an "OK" button in the Motif style. (p. 448)

  `motif-query-gadget` - A Motif style dialog box with multiple buttons. (p. 449)

- **Property sheet gadgets** (a Motif property sheet is pictured in figure 1-4)

  `prop-sheet` - Displays a set of labels and values and allows the values to be edited. This gadget can be easily displayed in its own window. (p. 411)

  `prop-sheet-for-obj` - A property sheet designed to display the slots in a Garnet object. (p. 415)

  `prop-sheet-with-OK` - A property sheet with OK-Cancel buttons. (p. 418) `prop-sheet-for-obj-with-OK` - A property sheet designed to display the slots in a Garnet object with attached OK-Cancel buttons. (p. 419)

`motif-prop-sheet-with-OK` - A property sheet with OK-Cancel buttons in the Motif style. (p. 452)

`motif-propt-sheet-for-obj-with-OK` - A Motif style property sheet designed to display the slots in a Garnet object with attached OK-Cancel buttons. (p. 453)

- **Scrolling windows**

`scrolling-window` - Supports a scrollable window (p. 391)

`scrolling-window-with-bars` - Scrolling window complete with scroll bars. (p. 391)

`motif-scrolling-window-with-bars` - Motif style scrolling window (p. 455)

- **Special gadgets**

`arrow-line` - A line with an arrowhead at one end (p. 395)

`double-arrow-line` - A line with arrowheads at both ends (p. 396)

`browser-gadget` - Used to examine structures and hierarchies (p. 397)

`graphics-selection` - Bounding boxes and interactors to move and change the size of other graphical objects. (p. 386)

`multi-graphics-selection` - Same as `graphics-selection`, but for multiple objects. (p. 388)

`polyline-creator` - For creating and editing polylines. (p. 401)

`MouseLine` and `MouseLinePopup` - A gadget that pops up a "help" string, informing the user about the object that the mouse is held over.

`standard-edit` - A module of predefined "cut" and "paste" procedures, and many other common editing functions. (p. 423)

1.2. Customization

The most important feature of the Garnet Gadgets is the ability to create a variety of interface styles from a small collection of prototype objects. Each gadget includes many parameters which may be customized by the designer, providing a great deal of flexibility in the behavior of the gadgets. The designer may, however, choose to leave many of the default values unchanged, while modifying only those parameters that integrate the object into the larger user interface.

The location, size, functionality, etc., of a gadget is determined by the values in each of its slots. When instances of gadgets are created, the instances inherit all of the slots and slot values from the prototype object except those slots which are specifically assigned values by the designer. The slot values in the prototype can thus be considered "default" values for the instances, which may be overridden when instances are created.[1] The designer may also add new slots not defined in the gadget prototype for use by special applications in the larger interface. Slot values may be changed after the instances are created by using the KR function `s-value`.

[1]See the KR manual for a more detailed discussion of inheritance.

1.3. Using Gadget Objects

The gadget objects reside in the `GARNET-GADGETS` package, which has the nickname "GG". We recommend that programmers explicitly reference the name of the package when creating instances of the gadgets, as in `garnet-gadgets:v-scroll-bar` or `gg:v-scroll-bar`. However, the package name may be dropped if the line `(use-package "GARNET-GADGETS")` is executed before referring to gadget objects.

Before creating instances of gadget objects, a set of component modules must be loaded. These modules are loaded in the correct order when the "-loader" files corresponding to the desired gadgets are used (see Chapter 2).

Since each top-level object is exported from the `GARNET-GADGETS` package, creating instances of gadget objects is as easy as instantiating any other Garnet objects. To use a gadget, an instance of the prototype must be defined and added to an interactor window. The following lines will display a vertical scroll bar in a window:

```
(create-instance ’MY-WIN inter:interactor-window
   (:left 0) (:top 0) (:width 300) (:height 500))
(create-instance ’MY-AGG opal:aggregate)
(s-value my-win :aggregate my-agg)
(create-instance ’MY-SCROLL-BAR garnet-gadgets:v-scroll-bar)
(opal:add-component my-agg my-scroll-bar)
(opal:update my-win)
```

The first two instructions create an interactor window named `my-win` and an aggregate named `my-agg`. The third instruction sets the `:aggregate` slot of `my-win` to `my-agg`, so that all graphical objects attached to `my-agg` will be shown in `my-win`. The next two instructions create an instance of the `v-scroll-bar` object named `my-scroll-bar` and add it as a component of `my-agg`. The last instruction causes `my-win` to become visible with `my-scroll-bar` inside.

In most cases, the use of a gadget will follow the same form as the preceding example. The important difference will be in the instantiation of the gadget object (the fifth instruction above), where slots may be given values that override the default values defined in the gadget prototype. The following example illustrates such a customization of the vertical scroll bar.

Suppose that we would like to create a vertical scroll bar whose values span the interval [0..30], with its upper-left coordinate at (25,50). This vertical scroll bar may be created by:

```
(create-instance ’CUSTOM-BAR garnet-gadgets:v-scroll-bar
   (:left 25)
   (:top 50)
   (:val-1 0)
   (:val-2 30))
```

This instruction creates an object called CUSTOM-BAR which is an instance of `v-scroll-bar`. The vertical scroll bar CUSTOM-BAR has inherited all of the slots that were declared in the `v-scroll-bar` prototype along with their default values, except for the coordinate and range values which have been specified in this schema definition (see section 3.1 for a list of customizable slots in the scroll bar objects).

1.4. Application Interface

There are several ways that the gadgets can interface with your application. This section describes several ways the you can get the gadgets to "do something" to your application.

1.4.1. The :value slot

In most gadgets, there is a top-level `:value` slot. This slot is updated automatically after a user changes the value or position of some part of the gadget. This is therefore the main slot through which the designer perceives action on the part of the user.

The `:value` slot may be accessed directly (by the KR functions `gv` and `gvl`) order to make other objects in the larger interface dependent on the actions of the user. The slot may also be set directly by the KR function `s-value` to change the current value or selection displayed by the gadget (except in the scrolling menu gadget, where the `:selected-ranks` slot must be set).

An instance of a gadget can be given initial values by setting the `:value` slot after the instance has been created. In most gadgets, this slot may not be given a value in the `create-instance` call, since this would override the formula in the slot. Therefore, the general procedure for selecting an initial value in a gadget is to create the instance, access the `:value` slot using `gv` (to initialize the formula in the slot and establish dependencies), and then use `s-value` to set the slot to the desired initial value.

See sections 5.1 and 5.4 for examples of the `:value` slot in use.

1.4.2. The :selection-function slot

In most gadgets there is a `:selection-function` slot which holds the name of a function to be called whenever the `:value` slot changes due to action by the user (such as the pressing of a button). The `:selection-function` is not automatically called when the designer's interface sets the `:value` slot directly.

This is probably the most important link between the gadgets and your application. By supplying a gadget with a selection function, then the gadget can execute some application-specific procedure when the user operates it.

In the scroll bars, sliders, trill device, and gauge, this function is called after the user changes the value by moving the indicator or typing in a new value (the function is called repeatedly while the user drags an indicator). In buttons and menus, it is called when the user changes the currently selected item or set of items, and it precedes the function attached locally to the item. In the labeled box, scrolling-input-string and scrolling-labeled-box, it is called after the user has finished editing the text (i.e., after a carriage return). In the `:graphics-selection` gadget, it is called whenever the user selects a new object or deselects the current object.

In the scrolling menu gadget, there are two selection functions, named `:scroll-selection-function` and `:menu-selection-function` which are called independently when the user moves the scroll bar or selects a menu item, respectively.

The function must take two parameters: the top-level gadget itself and the value of the top-level `:value` slot:

```
(lambda (gadget-object value))
```

In x-buttons, the parameter `value` will be a list of strings. The scrolling menu sends the menu item (a Garnet schema) on which the user just clicked as its `value`. Other gadgets will have only a single number or string as their `value`.

An example use of `:selection-function` is in section 5.2.

1.4.3. The :items slot

The button and menu gadgets are built up from items supplied by the designer. These items are supplied as a list in the `:items` slot of the gadgets. **Note:** Do not destructively modify the `:items` list; instead, create a new list using `list` or copy the old value with `copy-list` and modify the copy.

1.4.3.1. Item functions

There are several ways to specify items:

- **List of strings -** This is the obvious case, such as `'("Open" "Close" "Erase")`.
- **List of atoms -** In Garnet, the values of slots are often specified by atoms -- symbols preceded by a colon (e.g., `:center`). If a formula in the larger interface depends upon the `:value` slot of the button panel, then the designer may wish the items to be actual atoms rather than strings, so that the value is immediately used without being coerced. Such a list would look like `'(:left :center :right)`. The items will appear to the user as capitalized strings without colons.
- **List of objects -** In addition to string labels, the gadgets can have labels that are objects (like circles and rectangles). Such a list might look like `` `(,MY-CIRCLE ,MY-SQUARE ,OBJ3) ``. Objects, strings, and atoms can be mixed together in any `:items` list. Most of the demo functions for the gadgets use at least one object in the example.
- **List of label/function pairs -** This mode is useful when the designer wishes to execute a specific function upon selection of a button. If the `:items` slot contained the list `'(("Cut" My-Cut) ("Paste" My-Paste))`, then the function `My-Cut` would be executed when the button labeled "Cut" becomes selected. The designer must define these functions with two parameters:

  ```
  (lambda (gadget-object item-string))
  ```

  The *gadget-object* is the top-level gadget (such as a `text-button-panel`) and the *item-string* is the string (or atom) of the item that was just selected.

The item functions are executed along with the selection function whenever the user operates the gadget. These functions are different, however, because the selection function is executed when any item is selected, and the item functions are only executed when the item associated with them is selected.

The gadgets always assume that if an element of the `:items` list is a list, then the first element in the item is a label and the second element is a function. If you intend to use the `:items` list for storing application-specific data, you should avoid storing data in these reserved positions of the item elements. It is fine to store arbitrary data in the third and subsequent elements of an item list.

Section 5.3 shows an example implementation of item functions.

1.4.3.2. Adding and removing items

There are two ways to add and remove items from a button or menu gadget: use `add-item` and `remove-item` to change the `:items` list, or set the `:items` slot by hand using `s-value`. Both ways to change items are shown in the example below.

The various methods for changing items are

`opal:Add-Item` *gadget item* `[[:where]` *position[locator]* `[:key` *function-name*`]]` [*Method*]

`opal:Remove-Item` *gadget* `[`*item* `[:key` *function-name*`]]` [*Method*]

`opal:Remove-Nth-Item` *gadget n* [*Method*]

`opal:Change-Item` *gadget item n* [*Method*]

These methods are described in the Aggregadgets manual. `Add-item` will add *item* to the `:items` list of *gadget*, and will place it in the list according to the *position*, *locator*, and *key* parameters.

All gadgets that have an `:items` slot support `add-item` and the other methods (except for the `browser-gadget`, which has other item maintenance functions). The documentation for the `menubar` and `motif-menu` describes special features supported by those gadgets.

For example, consider adding an item to the X-BUTTONS-OBJ in the `x-button-panel` demo.

```
; Use opal:add-item in one step
(opal:add-item gg:X-BUTTONS-OBJ "newitem-1")

; Use s-value (directly or indirectly)
(push "newitem-2" (gv gg:X-BUTTONS-OBJ :items))
```

The `push` function uses `s-value` indirectly. `S-value` may also be used explicitly. After changing the `:items` list with `s-value`, the components of the gadget (like the individual buttons in a button panel) will be adjusted during the next call to `opal:update`. If information about the gadget (like its new dimensions) is required *before* the next update, the components can be adjusted manually with a call to `opal:notice-items-changed` with the gadget as a parameter. See the Aggregadgets Manual for more information about `opal:notice-items-changed`.

Because of internal references to the `:items` slot, destructive modification of the `:items` list is not allowed. If you change the list in the `:items` slot, you should create a new list (e.g., with `list`), or use `copy-list` on the original, and destructively modify the copy.

1.5. Constants with the Gadgets

At the top of most gadget definitions, there is a slot called `:maybe-constant` with a list of slots as its value. These are the slots that will be declared constant in an instance of a gadget, if the instance was created with its `:constant` slot set to T. By declaring a slot constant, the user promises that the value of that slot will never change, and all formulas that depend on it can be thrown away and replaced by absolute values.

Removing formulas that depend on constant slots can free up a large amount of storage space. Therefore, users who have finished designing part of an interface may want to go back through their gadget instances and delclare constant as many slots as possible.

In addition to using the special T value in a `:constant` list, you can selectively declare slots constant by listing them explicitly (e.g., `(:constant ’(:left :top))`). You can also use the `:except` keyword, as in the following schema:

```
(create-instance NIL gg:motif-radio-button-panel
   (:constant ’(T :except :active-p))
   (:left 10)(:top 30)
   (:items ’("Start" "Pause" "Quit")))
```

In this example, the user declares constant all of the slots in the `:maybe-constant` list, with the exception of `:active-p`. This allows the value of the `:active-p` slot to change, and retains all the formulas that depend on it (so that the gadget will update its appearance correctly when the value is toggled).

Constants are discussed in detail in the KR manual.

2. Accessing the Gadgets

2.1. Gadgets Modules

The schemata definitions in the gadgets package are modularized so that one schema may be used by several objects. For example, trill boxes with arrows pointing to the left and right are used in the horizontal scroll bar, the horizontal slider, and the trill device. As a result, all of the code for the gadget objects has a consistent style, and the gadgets themselves have a uniform look and feel.

2.2. Loading the Gadgets

Since much of the gadget code is shared by the top-level objects, a set of "parts" modules must be loaded before some of the top-level gadgets. The required modules are loaded in the proper order when the loader files corresponding to the desired gadgets are used. The standard gadgets and their associated loader files are listed in figure 2-1. The motif gadgets and loader files appear in figure 2-2. It is safe to load the "xxx-loader" files multiple times—they will not re-load the objects the second time.

To load the entire Gadget Set, execute `(load Garnet-Gadgets-Loader)` after loading the `Garnet-Loader`. *This is not recommended, since there are so many gadgets, and you will only need a few of them!* To load particular objects, such as the `v-slider` and `menu` gadgets, load the specific loader files:

```
(garnet-load "gadgets:v-slider-loader")
(garnet-load "gadgets:menu-loader")
```

For a discussion of the `garnet-load` function, see the Overview at the beginning of this reference manual.

2.3. Gadget Files

There are several gadgets files that normally have names that are longer than 31 characters. Since the Mac restricts the length of filenames to 31 characters, some gadget files have their names truncated on the Mac. Mac users may continue to specify the full-length names of these files by using `user::garnet-load`, described in the Overview section of this manual, which translates the regular names of the gadgets into their truncated 31-character names so they can be loaded. It is recommended that `garnet-load` be used whenever any Garnet file is loaded, so that typically long and cumbersome pathnames can be abbreviated by a short prefix.

2.4. Gadget Demos

Most gadgets have small demo functions that are loaded along with their schema definitions.[2] For example, after loading the `"v-slider-loader"`, you can do `gg:v-slider-go` to see a demo of the vertical slider.

A complete list of all gadget demos is included in the Demonstration Programs section of this reference manual. The names of all gadget demos are also mentioned at the top of each section in this Gadget manual.

[2] Unless the `:garnet-debug` key was removed from from the `*features*` list when the Garnet software was compiled or loaded (see the Hints manual).

`arrow-line` - "arrow-line-loader"
`browser-gadget` - "browser-gadget-loader"
`double-arrow-line` - "arrow-line-loader"
`error-gadget` - "error-gadget-loader"
`gauge` - "gauge-loader"
`graphics-selection` - "graphics-loader"
`h-scroll-bar` - "h-scroll-loader"
`h-slider` - "h-slider-loader"
`labeled-box` - "labeled-box-loader"
`load-gadget` - "save-gadget-loader"
`menu` - "menu-loader"
`menubar` - "menubar-loader"
`MouseLine` and `MouseLinePopup` - "mouseline-loader"
`multifont-gadget` - "multifont-loader"
`multi-graphics-selection` - "multi-selection-loader"
`option-button` - "option-button-loader"
`popup-menu-button` - "popup-menu-button-loader"
`prop-sheet` - "prop-sheet-loader"
`prop-sheet-for-obj` - "prop-sheet-loader"
`prop-sheet-for-obj-with-OK` - "prop-sheet-win-loader"
`prop-sheet-with-OK` - "prop-sheet-win-loader"
`query-gadget` - "error-gadget-loader"
`radio-button` - "radio-buttons-loader"
`radio-button-panel` - "radio-buttons-loader"
`save-gadget` - "save-gadget-loader"
`scrolling-input-string` - "scrolling-input-string-loader"
`scrolling-labeled-box` - "scrolling-labeled-box-loader".
`scrolling-menu` - "scrolling-menu-loader"
`scrolling-window` - "scrolling-window-loader"
`scrolling-window-with-bars` - "scrolling-window-loader"
`standard-edit` - "standard-edit-loader"
`text-button` - "text-buttons-loader"
`text-button-panel` - "text-buttons-loader"
`trill-device` - "trill-device-loader"
`v-scroll-bar` - "v-scroll-loader"
`v-slider` - "v-slider-loader"
`x-button` - "x-buttons-loader"
`x-button-panel` - "x-buttons-loader"

Figure 2-1: Loader files for Garnet Gadgets

`motif-check-button` - "motif-check-buttons-loader"
`motif-check-button-panel` - "motif-check-buttons-loader"
`motif-error-gadget` - "motif-error-gadget-loader"
`motif-gauge` - "motif-gauge-loader"
`motif-h-scroll-bar` - "motif-h-scroll-loader"
`motif-load-gadget` - "motif-save-gadget-loader")
`motif-menu` - "motif-menu-loader"
`motif-menubar` - "motif-menubar-loader"
`motif-option-button` - "motif-option-button-loader"
`motif-prop-sheet-...` - "motif-prop-sheet-win-loader"
`motif-query-gadget` - "motif-error-gadget-loader"
`motif-radio-button` - "motif-radio-buttons-loader"
`motif-radio-button-panel` - "motif-radio-buttons-loader"
`motif-save-gadget` - "motif-save-gadget-loader"
`motif-scrolling-labeled-box` - "motif-scrolling-labeled-box-loader"
`motif-scrolling-menu` - "motif-scrolling-menu-loader"
`motif-scrolling-window-with-bars` - "motif-scrolling-window-loader"
`motif-slider` - "motif-slider"
`motif-text-button` - "motif-text-buttons-loader"
`motif-text-button-panel` - "motif-text-buttons-loader"
`motif-trill-device` - "motif-trill-device-loader"
`motif-v-scroll-bar` - "motif-v-scroll-loader"

Figure 2-2: Loader files for Motif Gadgets

3. The Standard Gadget Objects

Each of the objects in the Gadget Set is an interface mechanism through which the designer obtains chosen values from the user. The scroll bars, sliders, gauge, and trill device all have a "continuous" flavor, and are used to obtain values between maximum and minimum allowed values. The buttons and menus are more "discrete", and allow the selection of a single choice from several alternatives.

The sections of this chapter describe the gadgets in detail. Each object contains many customizable slots, but the designer may choose to ignore most of them in any given application. If slot values are not specified when instances are created, then the default values will be used.

Each description begins with a list of the customizable slots and default values for the gadget object.

3.1. Scroll Bars

```
(create-instance ’gg:V-Scroll-Bar opal:aggregadget
   (:maybe-constant ’(:left :top :height :min-width :val-1 :val-2 :scr-trill-p
                      :page-trill-p :indicator-text-p :page-incr :scr-incr
                      :int-feedback-p :scroll-p :format-string :indicator-font
                      :visible))
   (:left 0)
   (:top 0)
   (:height 250)
   (:min-width 20)
   (:val-1 0)
   (:val-2 100)
   (:scr-incr 1)
   (:page-incr 5)
   (:scr-trill-p T)
   (:page-trill-p T)
   (:indicator-text-p T)
   (:int-feedback-p T)
   (:scroll-p T)
   (:indicator-font (opal:get-standard-font :fixed :roman :small))
   (:value (o-formula ...))
   (:format-string "~a")
   (:selection-function NIL)    ; (lambda (gadget value))
   )


(create-instance ’gg:H-Scroll-Bar opal:aggregadget
   (:maybe-constant ’(:left :top :width :min-height :val-1 :val-2 :scr-trill-p
                      :page-trill-p :indicator-text-p :page-incr :scr-incr
                      :int-feedback-p :scroll-p :format-string :indicator-font :visible))
   (:left 0)
   (:top 0)
   (:width 250)
   (:min-height 20)
   (:val-1 0)
   (:val-2 100)
   (:scr-incr 1)
   (:page-incr 5)
   (:scr-trill-p T)
   (:page-trill-p T)
   (:indicator-text-p T)
   (:int-feedback-p T)
   (:scroll-p T)
   (:indicator-font (create-instance NIL opal:font (:size :small)))
   (:value (o-formula ...))
   (:format-string "~a")
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

The loader file for the `v-scroll-bar` is "v-scroll-loader". The loader file for the `h-scroll-bar` is "h-scroll-loader".

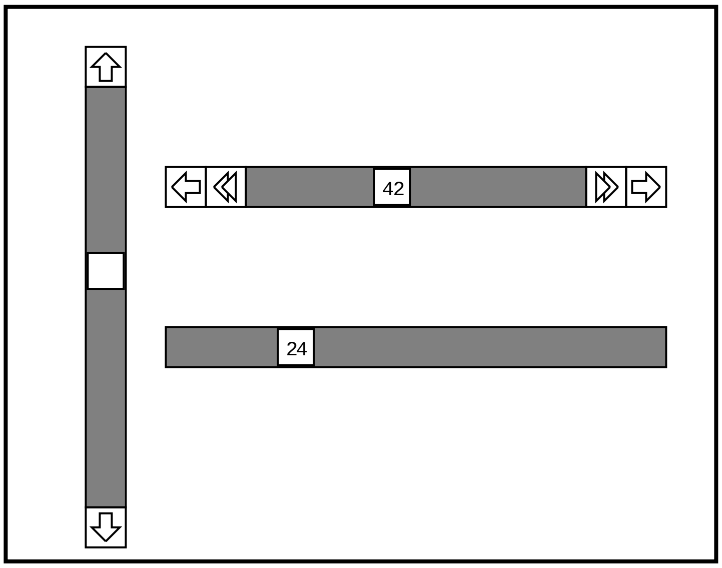


Figure 3-1: Vertical and horizontal scroll bars

The scroll bar is a common interface object used to specify a desired position somewhere in a range of possible values. The distance of the indicator from the top and bottom of its bounding box is a graphical representation of the currently chosen value, relative to the minimum and maximum allowed values.

The scroll bars in the Gadget Set, `v-scroll-bar` and `h-scroll-bar`, allow the interface designer to specify the minimum and maximum values of a range, while the `:value` slot is a report of the currently chosen value in the range. The interval is determined by the values in `:val-1` and `:val-2`, and either slot may be the minimum or maximum of the range. The value in `:val-1` will correspond to the top of the vertical scroll bar and the left of the horizontal scroll bar. The `:value` slot may be accessed directly by some function in the larger interface, and other formulas in the interface may depend on it. If the `:value` slot is set directly, then the appearance of the scroll bar will be updated accordingly.

The trill boxes at each end of the scroll bar allow the user to increment and decrement `:value` by precise amounts. The intent of the two sets of boxes is to give the user a choice between increment values -- either a conventional scroll of `:scr-incr` in the single arrow box or `:page-incr` in the double arrow box. There is no restriction on whether one value must be larger or smaller than the other.

In fact, the designer may choose to leave the trill boxes out completely. The slots `:scr-trill-p` and `:page-trill-p` may be set to NIL in order to prevent the appearance of the scroll boxes or page boxes, respectively.

The indicator may also be moved directly by mouse movements. Dragging the indicator while the left mouse button is pressed will cause a thick lined box to follow the mouse. The indicator then moves to the position of this feedback box when the mouse button is released. If `:int-feedback-p` is set to NIL, the thick lined box will not appear, and the indicator itself will follow the mouse. A click of the left mouse button in the background of the scroll bar will cause the indicator to jump to the position of the mouse.

With each change of the indicator position, the `:value` slot is updated automatically to reflect the new position. The current value is reported as a text string inside the indicator unless the slot `:indicator-text-p` is set to NIL.

Since the scroll bar must be wide enough to accommodate the widest text string in its range of values, the width of the vertical scroll bar (and similarly the height of the horizontal scroll bar) is the maximum of the width of the widest value and the `:min-width`. The `:min-width` will be used if there is no indicator text (i.e., `:indicator-text-p` is NIL), or if the `:min-width` is greater than the width of the widest value.

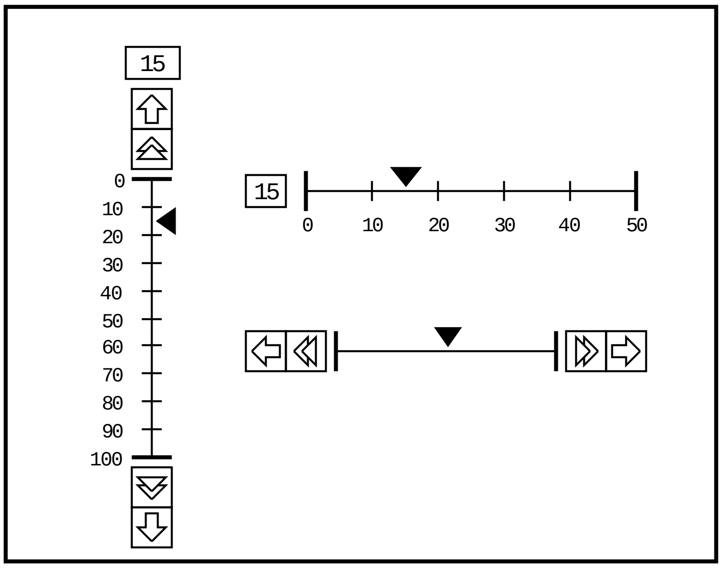
The slot `:scroll-p` is used to enable and disable the scrolling feature of the scroll bar. When `:scroll-p` is set to NIL, the trill boxes of the scroll bar become inactive and the background turns white. This ability to disable scrolling is useful in applications where the range of the scroll bar is not fixed. For example, in the `scrolling-menu` gadget, the scroll bar is disabled there are not enough items to fill the entire menu.

The font in which `:value` is reported in the indicator may be set in the slot `:indicator-font`.

3.2. Sliders

```
(create-instance 'gg:V-Slider opal:aggregadget
   (:maybe-constant '(:left :top :height :shaft-width :scr-incr :page-incr :val-1 :val-2
                      :num-marks :scr-trill-p :page-trill-p :tic-marks-p :enumerate-p
                      :value-feedback-p :scroll-p :value-feedback-font :enum-font
                      :format-string :enum-format-string :visible))
   (:left 0)
   (:top 0)
   (:height 250)
   (:shaft-width 20)
   (:scr-incr 1)
   (:page-incr 5)
   (:val-1 0)
   (:val-2 100)
   (:num-marks 11)
   (:scr-trill-p T)
   (:page-trill-p T)
   (:tic-marks-p T)
   (:enumerate-p T)
   (:value-feedback-p T)
   (:scroll-p T)
   (:value-feedback-font opal:default-font)
   (:enum-font (create-instance NIL opal:font (:size :small)))
   (:format-string "~a")
   (:enum-format-string "~a")
   (:value (o-formula ...))
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

```
(create-instance 'gg:H-Slider opal:aggregadget
   (:maybe-constant '(:left :top :width :shaft-height :scr-incr :page-incr :val-1 :val-2
                      :num-marks :tic-marks-p :enumerate-p :scr-trill-p :page-trill-p
                      :scroll-p :value-feedback-p :value-feedback-font :enum-font
                      :format-string :enum-format-string :visible))
   (:left 0)
   (:top 0)
   (:width 300)
   (:shaft-height 20)
   (:scr-incr 1)
   (:page-incr 5)
   (:val-1 0)
   (:val-2 100)
   (:num-marks 11)
   (:tic-marks-p T)
   (:enumerate-p T)
   (:scr-trill-p T)
   (:page-trill-p T)
   (:value-feedback-p T)
   (:scroll-p T)
   (:value-feedback-font opal:default-font)
   (:enum-font (create-instance NIL opal:font (:size :small)))
   (:format-string "~a")
   (:enum-format-string "~a")
   (:value (o-formula ...))
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```



The loader file for the `v-slider` is "v-slider-loader". The loader file for the `h-slider` is "h-slider-loader".

The `v-slider` and `h-slider` gadgets have the same functionality as scroll bars, but they are used when the context requires a different style. The slider is comprised of a shaft with perpendicular tic-marks and an indicator which points to the current chosen value. Optional trill boxes appear at each end of the slider, and the indicator can be moved with the same mouse commands as the scroll bar. The vertical slider has an optional feedback box above the shaft where the current value is displayed (this box is to the left of the horizontal slider). The value that appears in the feedback box may be edited directly by the user by pressing in the text box with the left mouse button and entering a new number.[3]

The slots `:value`, `:val-1`, `:val-2`, `:scr-incr`, `:page-incr`, `:scr-trill-p`, and `:page-trill-p` all have the same functionality as in scroll bars (see section 3.1).

The designer may specify the number of tic-marks to appear on the shaft in the slot `:num-marks`. This number includes the tic-marks at each end of the shaft in addition to the internal tic-marks. Tic-marks may be left out by setting the `:tic-marks-p` slot to NIL. If the slot `:enumerate-p` is set to T, then each tic-mark will be identified by its position in the range of allowed values. Also, numbers may appear without tic-marks marks by setting `:enumerate-p` to T and `:tic-marks-p` to NIL. The slot in which to specify the font for the tic-mark numbers is `:enum-font`.

The slot `:shaft-width` in the vertical slider (analogously, `:shaft-height` in the horizontal slider) is used to specify the width of the trill boxes at the end of the shaft. This determines the dimensions of the (invisible) bounding box for the interactors which manipulate the indicator.

The slot `:scroll-p` is used to enable and disable the scrolling feature of the sliders, just as in the scroll bars. When `:scroll-p` is set to NIL, the trill boxes of the slider become inactive, and the indicator ceases to move.

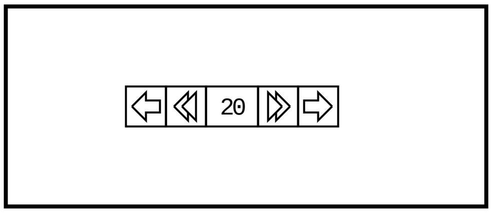
The font for the feedback of the current value (which appears at the end of the shaft) may be specified in `:value-feedback-font`. The value feedback may be left out completely by setting `:value-feedback-p` to NIL.

[3]Backspace and several editing commands are provided through Interactors. See "Text-Interactor" in the Interactors Manual.

The `:format-string` and `:enum-format-string` slots allow you to control the formatting of the text strings, in case the standard formatting is not appropriate. This is mainly useful for floating point numbers. The slots should each contain a string that can be passed to the lisp function `format`. The default string is `"~a"`.

3.3. Trill Device

```
(create-instance 'gg:Trill-Device opal:aggregadget
   (:maybe-constant '(:left :top :min-frame-width :min-height :scr-incr :page-incr
                      :val-1 :val-2 :scr-trill-p :page-trill-p :scroll-p
                      :value-feedback-p :format-string :value-feedback-font :visible))
   (:left 0)
   (:top 0)
   (:min-frame-width 20)
   (:min-height 20)
   (:scr-incr 1)
   (:page-incr 5)
   (:val-1 0) (:val-2 100)
   (:scr-trill-p T)
   (:page-trill-p T)
   (:scroll-p T)
   (:value-feedback-p T)
   (:value-feedback-font opal:default-font)
   (:value 20)
   (:format-string "~a")
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```



The loader file for the `trill-device` is "trill-device-loader".

The `trill-device` is a compact gadget which allows a value to be incremented and decremented over a range as in the scroll bars and sliders, but with only the numerical value as feedback. All slots function exactly as in horizontal sliders, but without the shaft and tic-mark features. As with sliders, the feedback value may be edited by the user.

A unique feature of the trill box is that either or both `:val-1` or `:val-2` may be NIL, implying no lower or upper bound on the input value, respectively. If numerical values for both slots are supplied, then clipping of the input value into the specified range occurs as usual. Otherwise, `:val-1` is assumed to be the minimum value, and clipping will not occur at the NIL endpoints of the interval.

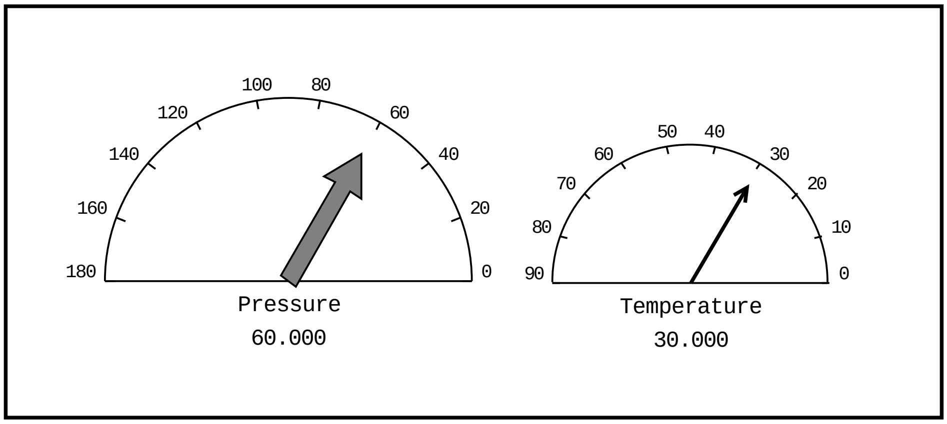
The width of the trill device may be either static or dynamic. If both `:val-1` and `:val-2` are specified, then the width of the value frame is the maximum of the widest allowed value and the `:min-frame-width`. Otherwise, the value frame will expand with the width of the value, while never falling below `:min-frame-width`.

The height of the trill device is the maximum of the greatest string height of all values in the range and the value of the slot `:min-height`. The `:min-height` will be used if there is no indicator text or if the `:min-height` is greater than the height of the tallest value.

The `:format-string` slot allows you to control the formatting of the text string, in case the standard formatting is not appropriate. This is mainly useful for floating point numbers. This slot takes a string that can be passed to the lisp function `format`. The default string is `"~a"`. For example:

```
(create-instance 'TRILL garnet-gadgets:trill-device
   (:left 35)(:top 70)(:val-1 0.0)(:val-2 1.0)(:scr-incr 0.01)
   (:page-incr 0.1)(:format-string "~4,2F"))
```

3.4. Gauge




```
(create-instance 'gg:Gauge opal:aggregadget
   (:maybe-constant '(:left :top :width :polygon-needle-p :int-feedback-p
                      :title :title-font :value-font :enum-font :num-marks
                      :tic-marks-p :enumerate-p :value-feedback-p :text-offset
                      :val-1 :val-2 :visible))
   (:left 0)
   (:top 0)
   (:width 230)
   (:val-1 0)
   (:val-2 180)
   (:num-marks 10)
   (:tic-marks-p T)
   (:enumerate-p T)
   (:value-feedback-p T)
   (:polygon-needle-p T)
   (:int-feedback-p T)
   (:text-offset 5)
   (:title "Gauge")
   (:title-font opal:default-font)
   (:value-font opal:default-font)
   (:enum-font (create-instance NIL opal:font (:size :small)))
   (:value (o-formula ...))
   (:format-string "~a")         ; How to print the feedback value
   (:enum-format-string "~a")   ; How to print the tic-mark values
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

The loader file for the `gauge` is "gauge-loader".

The `gauge` object is a semi-circular meter with tic-marks around the perimeter. As with scroll bars and sliders, this object allows the user to specify a value between minimum and maximum values. A needle points to the currently chosen value, and may either be a bare arrow or a thick, arrow-shaped polygon with a gray filling. The needle may be rotated by dragging it with the left mouse button pressed. Text below the gauge reports the current value to which the needle is pointing.

If the slot `:polygon-needle-p` is T, then the needle will be thick with a gray filling. If NIL, then the needle will be a bare arrow.

If `:int-feedback-p` is T, then the needle will not follow the mouse directly, but instead a short line will appear and be rotated. When the mouse button is released, the large needle will swing over to rest at the new location. The needle will follow the mouse directly if `:int-feedback-p` is set to NIL.

The slots `:num-marks`, `:tic-marks-p`, `:enumerate-p`, `:val-1`, `:val-2`, and `:enum-font` are implemented as in the sliders (see section 3.2). The value in `:val-1` corresponds to the right side of the gauge.

The title of the gauge is specified in `:title`. No title will appear if `:title` is NIL. The fonts for the title of the gauge and the current chosen value are specified in `:title-font` and `:value-font`, respectively.

If `:value-feedback-p` is T, then numerical text will appear below the gauge indicating the currently chosen value. The value in `:text-offset` determines the distance between the gauge and the title string, and between the title string and the value feedback.

The `:format-string` and `:enum-format-string` slots allow you to control the formatting of the text strings, in case the standard formatting is not appropriate. This is mainly useful for floating point numbers. The slots should each contain a string that can be passed to the lisp function `format`. The default string is `"~a"`.

3.5. Buttons

The button objects in the Garnet Gadgets can be either a single stand-alone button, or a panel of buttons. Each button in the set is related to the others by common interactors and constraints on both the sizes of the buttons and the text beside (or inside) the buttons.

The button objects all have several common features.

1. When used as a panel, the buttons are implemented with aggrelists, so all slots that can be customized in an aggrelist can be customized in the button panels.[4] These slots are:

   `:direction` — `:vertical` or `:horizontal` (default `:vertical`)

   `:v-spacing` — distance between buttons, if vertical orientation (default 5)

   `:h-spacing` — same, if horizontal orientation

   `:fixed-width-p` — whether all the buttons should have the width of the value in `:fixed-width-size`, or the width of each button should be determined by the width of the string associated with that button (default T)

   `:fixed-height-p` — same, but with heights

   `:fixed-width-size` — width of all components (default is the width of the widest button, as determined by the widest string)

   `:fixed-height-size` — same, but with heights

   `:h-align` — How to align buttons, if vertical orientation. Allowed values are `:left`, `:center`, or `:right`. (default `:right` for radio-buttons and x-buttons, `:center` for text-buttons)

   `:rank-margin` — after this many buttons, a new row (or column) will be started (default NIL)

   `:pixel-margin` — absolute position in pixels after which a new row (or column) will be started (default NIL)

[4] See the Aggregadgets manual for greater detail.

`:indent` — amount to indent the new row (or column) in pixels (default 0)

2. In the button and menu objects, the `:value` slot contains to the string or atom of the currently selected item (in the `x-button-panel` this value is a list of selected items). The currently selected object is named in the `:value-obj` slot. In order to set an item to be selected, either the `:value` slot of the button panel must be set with the desired string or atom from the `:items` list, or the `:value-obj` slot must be set with the desired button object (see section 5.4 for examples of selecting buttons).

3. The `:width` of the buttons is determined by the width of the longest item, and therefore cannot be specified by the designer. However, the `:width` is computed internally and may be accessed after the object is instantiated. (The :height is computed similarly.)

4. The shadow below each button has the effect of simulating a floating three-dimensional button. When the left mouse button is clicked on one of the gadget buttons, the button frame moves onto the shadow and appears to be depressed. The slot `:shadow-offset` specifies the amount of shadow that appears under the button when it is not pressed. A value of zero implies that no shadow will appear (i.e., no floating effect).

5. There is a gray border in the frame of each of the buttons, the width of which may be specified in the slot `:gray-width`.

6. The strings or atoms associated with each button are specified in the `:items` slot. See section 1.4.3 for a discussion of specifying items and item functions.

7. The font in which the button labels appear may be specified in the `:font` slot.

8. Most of the buttons and button panels have a `:toggle-p` slot. When the value of this slot is T, then the button will become deselected if it is clicked a second time. Otherwise, after the button is selected the first time, it is always selected (though its `:selection-function` and associated item functions will continue to be executed each time it is pressed.

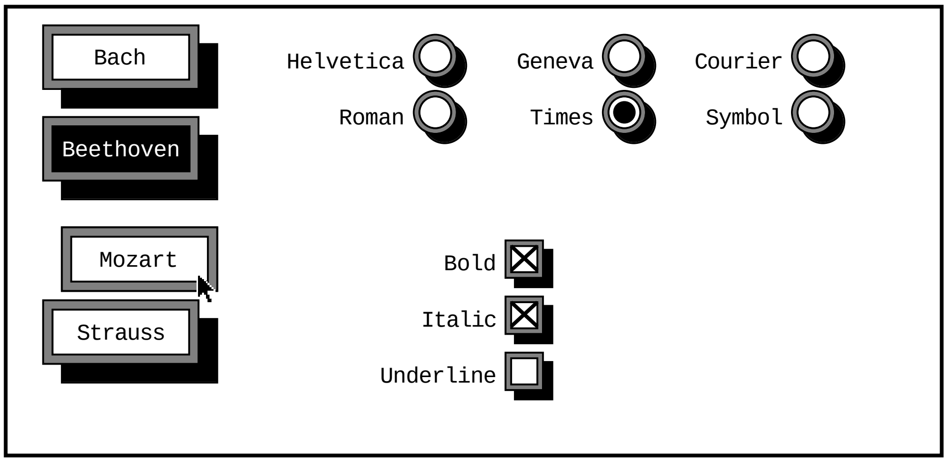


Figure 3-2: Text buttons, radio buttons, and x-buttons

3.5.1. Text Buttons

```
(create-instance 'gg:Text-Button opal:aggregadget
   (:maybe-constant '(:left :top :shadow-offset :text-offset :gray-width
                      :string :toggle-p :font :final-feedback-p :visible))
   (:left 0)
   (:top 0)
   (:shadow-offset 10)
   (:text-offset 5)
   (:gray-width 5)
   (:string "Text Button")
   (:toggle-p T)
   (:font opal:default-font)
   (:final-feedback-p T)
   (:value (o-formula (if (gvl :selected) (gvl :string))))
   (:selected (o-formula (gvl :value)))   ; This slot is set by the interactor
   (:selection-function NIL)   ; (lambda (gadget value))
   )


(create-instance 'gg:Text-Button-Panel opal:aggregadget
   (:maybe-constant '(:left :top :direction :v-spacing :h-spacing :h-align
                      :fixed-width-p :fixed-width-size :fixed-height-p
                      :fixed-height-size :indent :rank-margin :pixel-margin
                      :shadow-offset :text-offset :gray-width :final-feedback-p
                      :toggle-p :font :items :visible))
   (:left 0)
   (:top 0)
   (:shadow-offset 10)
   (:text-offset 5)
   (:gray-width 5)
   (:final-feedback-p T)
   (:toggle-p NIL)
   (:font opal:default-font)
   (:items '("Text 1" "Text 2" "Text 3" "Text 4"))
   (:value-obj NIL)
   (:value (o-formula (gvl :value-obj :string)))
   (:selection-function NIL)   ; (lambda (gadget value))
   <All customizable slots of an aggrelist>)
```

The loader file for the `text-button` and `text-button-panel` is "text-buttons-loader".

The `text-button-panel` object is a set of rectangular buttons, with the string or atom associated with each button centered inside. When a button is pressed, the text of the button will appear in inverse video if `:final-feedback-p` is T. The `text-button` is just a single button.

The distance from the beginning of the longest label to the inside edge of the button frame is specified in `:text-offset`. The value in `:text-offset` will affect the height and width of every button when specified.

3.5.2. X Buttons

```
(create-instance 'gg:X-Button opal:aggregadget
   (:maybe-constant '(:left :top :button-width :button-height
                      :shadow-offset :text-offset :gray-width
                      :text-on-left-p :toggle-p :string :font :visible))
   (:left 0)
   (:top 0)
   (:button-width 20)
   (:button-height 20)
   (:shadow-offset 5)
   (:text-offset 5)
   (:gray-width 3)
   (:text-on-left-p T)
   (:string "X Button")
   (:toggle-p T)
   (:font opal:default-font)
   (:value (o-formula (if (gvl :selected) (gvl :string))))
   (:selected (o-formula (gvl :value))) ; Set by interactor
   (:selection-function NIL)             ; (lambda (gadget value))
   )


(create-instance 'gg:X-Button-Panel opal:aggregadget
   (:maybe-constant '(:left :top :direction :v-spacing :h-spacing :h-align
                      :fixed-width-p :fixed-width-size :fixed-height-p :fixed-height-size
                      :indent :rank-margin :pixel-margin :button-width :button-height
                      :shadow-offset :text-offset :gray-width :text-on-left-p
                      :font :items :visible))
   (:left 0)
   (:top 0)
   (:button-width 20)
   (:button-height 20)
   (:shadow-offset 5)
   (:text-offset 5)
   (:gray-width 3)
   (:text-on-left-p T)
   (:font opal:default-font)
   (:items '("X-label 1" "X-label 2" "X-label 3"))
   (:value-obj NIL)
   (:value (o-formula (mapcar #'(lambda (object)
                                   (gv object :string))
                              (gvl :value-obj))))
   (:selection-function NIL)  ; (lambda (gadget value))
   <All customizable slots of an aggrelist>)
```

The loader file for the `x-button` and `x-button-panel` is "x-buttons-loader".

The `x-button-panel` object is also a set of rectangular buttons, but the item associated with each button appears either to the left or to the right of the button. Any number of buttons may be selected at one time, and clicking on a selected button de-selects it. Currently selected buttons are graphically indicated by the presence of a large "X" in the button frames. The `x-button` is just a single button.

Since the `x-button-panel` allows selection of several items at once, the `:value` slot is a list of strings (or atoms), rather than a single string. Similarly, `:value-obj` is a list of objects.

The slot `:text-on-left-p` specifies whether the text will appear on the right or left of the x-buttons. A NIL value indicates the text should appear on the right. When text appears on the right, the designer will probably want to set `:h-align` to `:left` in order to left-justify the text against the buttons.

The distance from the labels to the buttons is specified in `:text-offset`.

The slots `:button-width` and `:button-height` specify the width and height of the x-buttons. The "X" will stretch to accommodate these dimensions.

3.5.3. Radio Buttons

```
(create-instance 'gg:Radio-Button opal:aggregadget
   (:maybe-constant '(:left :top :button-diameter :shadow-offset :text-offset
                      :gray-width :string :text-on-left-p :toggle-p :font :visible))
   (:left 0) (:top 0)
   (:button-diameter 23)
   (:shadow-offset 5) (:text-offset 5) (:gray-width 3)
   (:string "Radio button")
   (:toggle-p T)
   (:text-on-left-p T)
   (:font opal:default-font)
   (:value (o-formula (if (gvl :selected) (gvl :string))))
   (:selected (o-formula (gvl :value))) ; Set by interactor
   (:selection-function NIL)            ; (lambda (gadget value))
   )
```

```
(create-instance 'gg:Radio-Button-Panel opal:aggregadget
   (:maybe-constant '(:left :top :direction :v-spacing :h-spacing :h-align
                      :fixed-width-p :fixed-width-size :fixed-height-p :fixed-height-size
                      :indent :rank-margin :pixel-margin :button-diameter :shadow-offset
                      :text-offset :gray-width :text-on-left-p :toggle-p :font
                      :items :visible))
   (:left 0)
   (:top 0)
   (:button-diameter 23)
   (:shadow-offset 5)
   (:text-offset 5)
   (:gray-width 3)
   (:text-on-left-p T)
   (:toggle-p T)
   (:font opal:default-font)
   (:items '("Radio-text 1" "Radio-text 2" "Radio-text 3" "Radio-text 4"))
   (:value-obj NIL)
   (:value (o-formula (gvl :value-obj :string)))
   (:selection-function NIL)    ; (lambda (gadget value))
   <All customizable slots of an aggrelist>)
```

The loader file for the `radio-button` and `radio-button-panel` is "radio-buttons-loader".

The `radio-button-panel` is a set of circular buttons with items appearing to either the left or right of the buttons (implementation of `:text-on-left-p` and `:text-offset` is identical to x-buttons). Only one button may be selected at a time, with an inverse circle indicating the currently selected button. A `radio-button` is a single button.

3.6. Option Button

```
(create-instance 'gg:Option-Button opal:aggregadget
  (:maybe-constant '(:left :top :text-offset :label :button-offset :button-shadow-offset
                     :items :initial-item :button-font :label-font :button-fixed-width-p
                     :v-spacing :keep-menu-in-screen-p :menu-h-align))
  (:left 40) (:top 40)
  (:text-offset 4)
  (:label "Option button:")
  (:button-offset 10)
  (:button-shadow-offset 5)
  (:items '("Item 1" "Item 2" "Item 3" "Item 4"))
  (:initial-item (o-formula (first (gvl :items))))
  (:button-font opal:default-font)
  (:label-font (opal:get-standard-font NIL :bold NIL))
  (:value (o-formula (gvl :option-text-button :string)))
  (:button-fixed-width-p T)
  (:v-spacing 0)
  (:menu-h-align :left)
  (:keep-menu-in-screen-p T)
  (:selection-function NIL)    ; (lambda (gadget value))
  ...)
```

The loader file for the `option-button` is "option-button-loader".

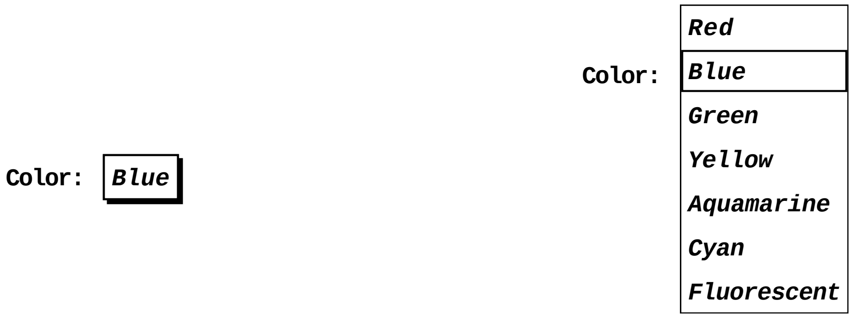


Figure 3-3: An option button in its normal state (left), and showing the available options after the button is pressed (right).

When the left mouse button is clicked and held on an option button, a menu will pop up, from which items can be selected by moving the pointer to the desired item and releasing the mouse button. Figure 3-3 shows an option button in its normal state (on the left) and when the button is pressed.

The `:items` slot is a list of strings or Garnet objects, which will appear in the menu. The `:initial-item` slot contains the initial item that will appear in the button. This slot MUST be non-NIL, and should contain either an element of the `:items` list, or a formula to calculate the same.

The `:text-offset` slot specifies how far from the frame the text should begin. The slot `:button-offset` specifies how far from the label the button should begin. The `:button-shadow-offset` contains the size of the button's shadow.

The `:label` slot contains a string that appears before the option button. If no label is desired, this slot can be set to the empty string, "".

The `:button-font` and `:label-font` slots specify the fonts to use in the button and the label. The font of the items in the menu is the same as the font in the `:button-font` slot.

The `:value` slot contains the currently selected item, which is the same as the value in the `:string` slot of the button.

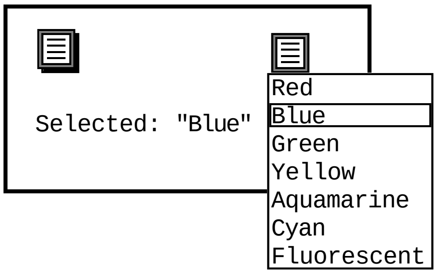
The `:button-fixed-width-p` slot specifies whether to keep the button's width constant or not. If it is set to T, the button's width will be the width of the longest string in the `:items` slot. If it is set to NIL, the width of the button will be the width of the currently selected item.

The value in `:v-spacing` specifies the amount of space between each menu item.

The `:menu-h-align` slot should be either `:left`, `:center`, or `:right`, and specifies the justification of the menu items.

If the `:keep-menu-in-screen-p` slot is T, then the menu will never pop out of the screen, i.e., the top of the menu will never be less than the screen's top, and the bottom of the menu will never be greater than the screen's bottom. If this slot is set to NIL, the menu may pop out of the top or out of the bottom of the screen. NOTE: If the number of items in the menu makes it so that both the top of the menu and the bottom of the menu are out of the screen, this slot will be disregarded.

3.7. Popup-Menu-Button

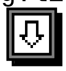



```
(create-instance ’gg:Popup-Menu-Button gg:text-button
  (:left 0)
  (:top 0)
  (:string gg:lines-bitmap)
  (:items ’("Item 1" "Item 2" "Item 3" "Item 4"))
  (:v-spacing 0)
  (:h-align :LEFT)
  (:item-font opal:default-font)
  (:item-to-string-function
   #’(lambda (item)
       (if item
           (if (or (stringp item) (schema-p item))
               item
               (string-capitalize (string-trim ":" item)))
           "")))
  (:min-menu-width 0)
  (:shadow-offset 2)
  (:text-offset 3)
  (:gray-width 2)
  (:selection-function NIL)    ; (lambda (gadget value))
  (:value (o-formula ...))
  (:position :below)
  (:keep-menu-in-screen-p T)
```

The loader file for the `popup-menu-button` is `popup-menu-button-loader`, and you can see a demo by executing `(gg:popup-menu-button-go)`. (Sorry, there isn’t a Motif version yet.)

This is a combination of a button and a popup menu. When you press on the button, the menu is shown, and then you can select something from the menu, and then the menu disappears. If you release outside of the menu, the menu just goes away and keeps its current value. The button itself can show a string or an arbitrary Garnet object (e.g., a bitmap, as shown here).

The `:left` and `:top` determine when the button goes.

The `:string` slot determines what is shown in the button. It can be a regular string (e.g., `"Value"`) or an arbitrary Garnet object. The default value is the `gg:lines-bitmap` shown above. Another bitmap provided is `gg:downarrow-bitmap` which looks like .

The `:items` slot holds the items that are shown in the popup menu. It can have the standard format for items (e.g., a list of strings, objects, pairs of strings and functions, etc.). See section 1.4.3 for more information.

The `:v-spacing`, `:h-align`, and `:item-font` control the display of the menu items. See the Gadgets manual for menus for more information.

The `:min-menu-width` slot can contain a minimum width for the popup menu. You might use this to make the menu line up with a text entry field.

The `:item-to-string-function` can be used to convert the values in the `:items` list into strings.

The `:shadow-offset`, `:text-offset` and `:gray-width` parameters control the appearance of the button itself.

When the user selects a menu item, the `:selection-function` is called with parameters: `(lambda (gadget value))`, where the gadget is the popup-menu-button and the value is the appropriate item from `:items`. The `:value` slot will also be set with the appropriate item.

The position of the menu with respect to the button is controlled by the `:position` parameter. Legal options are:

`:below` - the menu is below and left justified with the button (the default).

`:left` - the menu will be centered vertically at the left of the button.

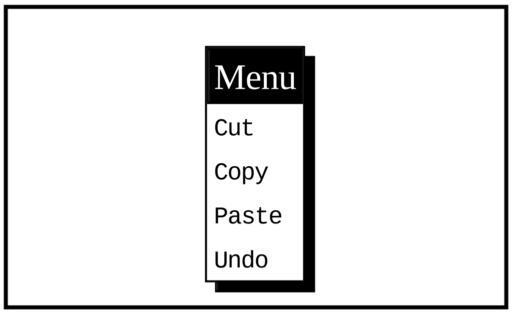
`:right` - the menu will be centered vertically at the right of the button.

a list of two numbers (x y) - the menu will be at this location. The `:position` slot can contain a formula that calculates these numbers.

If `:keep-menu-in-screen-p` is non-NIL, then the position computed based on the `:position` argument will be adjusted so the menu always stays in the screen. Otherwise, the menu might extend off the screen edges.

3.8. Menu

```
(create-instance 'gg:Menu opal:aggregadget
   (:maybe-constant '(:left :top :v-spacing :h-align :shadow-offset
                      :text-offset :title :title-font :items :item-font
                      :item-to-string-function :visible))
   (:left 0)
   (:top 0)
   (:v-spacing 0)
   (:h-align :left)
   (:shadow-offset 5)
   (:text-offset 4)
   (:min-menu-width 0)
   (:title NIL)
   (:title-font (create-instance NIL opal:font
                   (:family :serif)
                   (:size :large)
                   (:face :roman)))
   (:items '("Item 1" "Item 2" "Item 3" "Item 4"))
   (:item-font opal:default-font)
   (:item-to-string-function #'(lambda (item)
                                 (if item
                                     (if (or (stringp item) (schema-p item))
                                         item
                                       (string-capitalize (string-trim ":" item)))
                                   "")))
   (:selection-function NIL)    ; (lambda (gadget value))
   (:value-obj NIL)
   (:value (o-formula (gvl :value-obj :string))))
```



The loader file for the menu is "menu-loader".

The menu object is a set of text items surrounded by a rectangular frame, with an optional title above the items in inverse video. When an item is selected, a box is momentarily drawn around the item and associated item functions and global functions are executed.

The :items slot may be a list of strings, atoms, string/function pairs or atom/function pairs, as with buttons (see section 3.5). If this slot is s-value'd with a new list of items, the components of the gadget will be adjusted automatically during the next call to opal:update.

The amount of shadow that appears below the menu frame (the menu frame is stationary) is specified in :shadow-offset. A value of zero implies that no shadow will appear.

The slot :h-align determines how the menu items are justified in the frame. Allowed values are :left, :center and :right.

The slot :text-offset is the margin spacing -- the distance from the frame to the longest string.

The slot :item-font determines the font in which the items will appear.

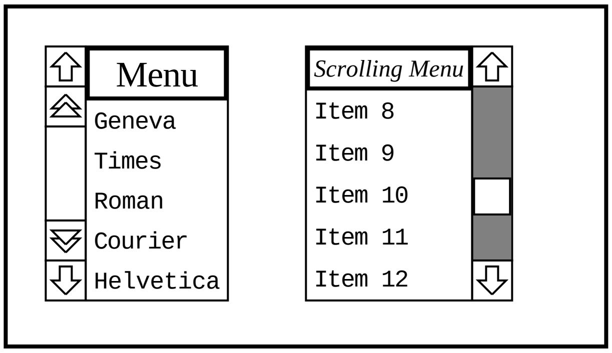
A title for the menu may be specified as a string in the :title slot. If :title is NIL, then no title will appear. The font in which the title should appear is specified in :title-font.

The :items slot may be a list of any objects, including strings, atoms, schemas, string/function pairs, etc. The default scrolling menu assumes that :items contains a list as described in section 3.5, but this can be easily changed by the designer. A function defined in :item-to-string-function takes an item (or the first element of an item pair) and returns a string corresponding to that item for display in the menu. The default function for this slot is

```
(lambda (item)
  (if item
      (if (stringp item)
          item
          (string-capitalize (string-trim ":" item)))
      ""))
```

This function takes an item and returns it if it is a string, or coerces it into a string if it was an atom. See section 5.5 for an example where the :items list is composed of Garnet schemas.

3.9. Scrolling Menu




```
(create-instance 'gg:Scrolling-Menu opal:aggregadget
   (:maybe-constant '(:left :top :scroll-on-left-p :min-scroll-bar-width :scr-trill-p
                      :page-trill-p :indicator-text-p :scr-incr :page-incr
                      :int-scroll-feedback-p :indicator-font :min-frame-width :v-spacing
                      :h-align :multiple-p :items :item-to-string-function :item-font
                      :num-visible :int-menu-feedback-p :final-feedback-p :text-offset
                      :title :title-font :visible))
   (:left 0) (:top 0)

   ;; Scroll bar slots
   (:scroll-on-left-p T)
   (:min-scroll-bar-width 20)
   (:scr-trill-p T)
   (:page-trill-p T)
   (:indicator-text-p NIL)
   (:scr-incr 1)
   (:page-incr 5)
   (:int-scroll-feedback-p NIL)
   (:indicator-font (create-instance NIL opal:font (:size :small)))
   (:scroll-selection-function NIL)

   ;; Menu slots
   (:min-frame-width 0)
   (:v-spacing 6)
   (:h-align :left)
   (:multiple-p T)
   (:toggle-p T)
   (:items '("Item 1" "Item 2" "Item 3" ... "Item 20"))
   (:item-to-string-function
       #'(lambda (item)
           (if item
               (if (stringp item)
                   item
                   (string-capitalize (string-trim ":" item)))
               "")))
   (:item-font opal:default-font)
   (:num-visible 5)
   (:int-menu-feedback-p T)
   (:final-feedback-p T)
   (:text-offset 4)
   (:title NIL)
   (:title-font (create-instance NIL opal:font
                   (:family :serif)
                   (:size :large)
                   (:face :roman)))
   (:menu-selection-function NIL)
   (:selected-ranks NIL)
   (:value (o-formula ...)))
```

The loader file for the `scrolling-menu` gadget is "scrolling-menu-loader".

The `scrolling-menu` object is a combination of a vertical scroll bar and a menu which allows the user to only see a subset of the available choices in a menu at one time. The set of visible choices is changed by moving the scroll bar, which causes the choices to scroll up or down in the menu.

3.9.1. Scroll Bar Control

If the slot `:scroll-on-left-p` is T, then the scroll bar will appear on the left side of the menu. Otherwise, the scroll bar will be on the right.

The slot `:min-scroll-bar-width` determines the minimum width of the scroll bar. The scroll bar will be wider than this width only if the indicator text is too wide to fit into this width.

The interim feedback of the scroll bar is controlled by the slot `:int-scroll-feedback-p`. If this slot is set to T, then a thick-lined box will follow the mouse when the user drags the indicator. Otherwise, the indicator will follow the mouse directly.

A function may be specified in `:scroll-selection-function` to be executed whenever the user changes the scroll bar, either by clicking on the trill boxes or by dragging the indicator. The function takes the same parameters as the usual selection function described in section 1.4.2.

The slots `:scr-trill-p`, `:page-trill-p`, `:scr-incr`, `:page-incr`, `:indicator-text-p`, and `:indicator-font` are all used for the scroll bar in the scrolling menu in the same way as the vertical scroll bar described in section 3.1.

3.9.2. Menu Control

The minimum width of the scrolling menu frame is determined by `:min-frame-width`. The scrolling menu will appear wider than this value only if the title or the longest item string will not fit in a menu of this width.

The `:v-spacing` slot determines the distance between each item in the menu.

The justification of the items in the menu is determined by the slot `:h-align` which may be either `:left`, `:center`, or `:right`.

If the value of `:multiple-p` is T, then the user may make multiple selections in the menu by clicking on items while holding down the shift key. If this slot is NIL, then only single selections are permitted.

The `:toggle-p` slot specifies whether to toggle the current selection when it is clicked on again. If `:toggle-p` is NIL, then a selected item can be clicked upon for any number of times and it will stay selected. If the `:toggle-p` slot is set to T (the default), clicking on an already selected item will cause the item to become unselected. NOTE: Clicking on a selected item while doing multiple selections will always toggle the selection, regardless of the value of the `:toggle-p` slot.

The `:item-to-string-function` slot is identical in operation to the one described for the `gg:menu` in section 3.8. If the `:items` slot does not contain a list of the usual items or item/function pairs, then this function should return the conversion of each element into a valid item. The default `:item-to-string-function` assumes that the `:items` list is composed of the usual items or item/function pairs.

The slot `:num-visible` determines how many items should be visible in the menu at one time.

A box will appear around the item being selected while the mouse button is held down if the slot `:int-menu-feedback-p` is T.

Selected items will appear in inverse video if the slot `:final-feedback-p` is set to T.

The slot `:text-offset` determines the distance from each string to the menu frame.

A title will appear above the menu if a title string is specified in `:title`. If `:title` is NIL, then no title will appear. The font of the title is in `:title-font`.

The font of the items is in `:item-font`.

The `:selected-ranks` slot is used by the designer to select items in the menu. The slot contains a list of indices which correspond to the ranks of the selected items in the `:items` list. The ranks are zero-based. For example, if the `:selected-ranks` slot contained `'(0 3)`, then the first and fourth items (not necessarily visible) in the scrolling menu will be selected.

A function defined in `:menu-selection-function` will be executed whenever the user selects an item from the menu. This function takes two parameters,

```
(lambda (gadget scrolling-menu-item))
```

where *gadget* is the programmer's instance of the `scrolling-menu` and *scrolling-menu-item* is the object just selected by the user. The item associated with the user's selection can be obtained through the `:item` slot of the *scrolling-menu-item*:

```
(gv scrolling-menu-item :item) --> The item just selected
```

3.10. Menubar

```
(create-instance 'gg:Menubar opal:aggrelist
   (:left 0)(:top 0)
   (:items NIL)
   (:title-font (create-instance NIL opal:font (:size :large)))
   (:item-font (create-instance NIL opal:font (:face :italic)))
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

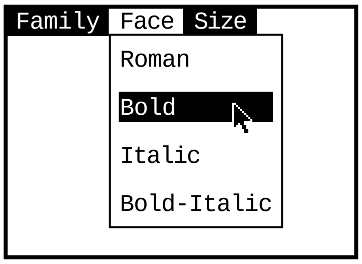


Figure 3-4: Picture of a pull-down menu (an instance of `menubar`)

The `menubar` gadget is a set of pull down menus that is similar to the Macintosh design. When the user clicks on an inverse bar item, a submenu pops up and the user can then choose one of the displayed items.

NOTE: There is no `:value` slot in this gadget. The designer should define functions in the `:selection-function` or `:items` slots to propagate the user's selections to the rest of the interface (see below).

The complete `menubar` gadget is a collection of three objects. In addition to the top-level `menubar` object, there are `bar-item` and `submenu-item` objects. The `menubar` is an aggrelist of `bar-item` objects, which are the inverse-video text objects that appear horizontally at the top of the window. The `submenu-item` objects are vertically arranged in an aggrelist within each `bar-item`.

The programmer may approach the `menubar` from two perspectives: the traditional Garnet way which involves setting the `:items` slot and allowing the gadget to maintain its own components, or from a bottom-up approach which involves creating the sub-objects and manually adding (and removing) them from the `menubar` instance.

Programmers who choose the Garnet approach can ignore most of the functions described below, since interaction with the `menubar` will almost exclusively involve setting the `:items` slot. The other approach requires creating instances of `bar-item` and `submenu-item` gadgets and adding them as components to a `menubar` using the support functions.

3.10.1. Item Selection Functions

There are three levels of functions in the `menubar` gadget that may be called when the user makes a selection. Functions can be attached to individual submenu-items, whole submenus, or the top-level `menubar`. All three types of functions take the parameters

```
(lambda (gadget menu-item submenu-item))
```

When a function is supplied in the `:selection-function` slot of the `menubar`, it will be executed whenever any item is selected from any of the submenus. If a function is attached to a submenu (e.g., it is the value for *m1func* in the `:items` syntax of section 3.10.2), then it is executed when any item is chosen from that submenu. If a function is attached to a submenu-item (e.g., *mX,Yfunc*), then it is executed only when that submenu-item is selected.

The order for calling these functions is: first, the submenu function is called, then the submenu-item function is called, and finally the `:selection-function` is called.

3.10.2. Programming the Menubar in the Traditional Garnet Way

The `:items` slot of the `menubar` is a complicated list with the following format:

```
(:items '(("m1" m1func (("m1,1" [m1,1func])...("m1,N" [m1,Nfunc])))
          ("m2" m2func (("m2,1" [m2,1func])...("m2,N" [m2,Nfunc])))
          ...))
```

where *"mN"* is a string or atom that is the title of a menu (atoms appear as capitalized strings in the submenu titles), *"mX,Y"* is a string or atom in menu X, row Y, *mNfunc* is executed when any item in menu N is selected, and *mX,Yfunc* is executed when item *"mX,Y"* is selected. See section 3.10.1 for the parameters of these functions. NOTE: the syntax above requires that the submenu-items be described with lists, even when no submenu-item functions are supplied (i.e., the list `("m1,1")` is required instead of just the string `"m1,1"`).

In order to maintain the syntax of the sublists, the submenu functions (*m1func* and *m2func* above) must always be supplied. Thus, NIL should be placed in this position if no function is to be executed for the submenu. The submenu-item functions (*m1,1func* etc. above) are optional and may be omitted.

The `:title-font` is the font for the `bar-item` objects which appear in inverse video, and the `:item-font` is the font for the `submenu-item` objects arranged vertically in the pop-up menus.

3.10.2.1. An example

```
(create-instance 'WIN inter:interactor-window
   (:left 750)(:top 80)(:width 200)(:height 200)
   (:title "Menubar Example"))
(s-value WIN :aggregate (create-instance 'TOP-AGG opal:aggregate))
(opal:update WIN)

(defun Fixed-Fn (gadget menu-item submenu-item)
  (format t "Calling Fixed-Fn with ~S ~S ~S.~%" gadget menu-item submenu-item))

(defun Face-Fn (gadget menu-item submenu-item)
  (format t "Calling Face-Fn with ~S ~S ~S.~%" gadget menu-item submenu-item))

(create-instance 'DEMO-MENUBAR garnet-gadgets:menubar
   (:items
    '(("family" NIL
       (("fixed" Fixed-Fn)("serif")("sans-serif")))
      ("face" Face-Fn
       (("roman")("bold")("italic")("bold-italic")))
      ("size" NIL
       (("small")("medium")("large")("very-large"))))))

(opal:add-component TOP-AGG DEMO-MENUBAR)
(opal:update WIN)
```

Figure 3-5: The code to generate the picture in Figure 3-4

The code in Figure 3-5 creates the `menubar` picture shown in Figure 3-4. It illustrates the Garnet method for handling the `menubar` gadget.

3.10.2.2. Adding items to the menubar

There are two types of items that can be added to a `menubar`: an entire submenu can be added to the top-level `menubar`, or single submenu-item can be added to a submenu.

The `add-item` method for the `menubar` can be used to add submenus --

`opal:Add-Item` *menubar submenu* `[[:where]` *position [locator]* `[:key` *index-function*`]]` [*Method*]

Using the standard Garnet method, the *submenu* parameter should be a sublist of a top-level items list, (e.g., `'("underline" NIL (("none") ("single") ("double")))`. The remaining optional parameters follow the standard `add-item` definition described in the Aggregadgets manual, and refer to the placement of the new submenu among the existing submenus. *Locator* should be some element of the current *:items* list, or may be the title of a submenu when the *index-function* is `#'car` (see examples below).

For example, each of the following lines will add a new submenu to DEMO-MENUBAR in Figure 3-5:

```
(opal:add-item DEMO-MENUBAR '("font-name" NIL (("courier") ("times") ("geneva"))))
(opal:add-item DEMO-MENUBAR
               '("other-fonts" NIL (("helvetica") ("chicago")))
               :after '("family" NIL (("fixed" Fixed-Fn)("serif")("sans-serif"))))
(opal:add-item DEMO-MENUBAR
               '("symbols" NIL (("mathematical") ("greek")))
               :before "face" :key #'car)
```

Individual submenu-items can be added to a `menubar` with the following function:

`add-submenu-item` *menubar submenu-title submenu-item* `[[:where]` *position [locator]* `[:key` *index-function*`]])`

This function adds the new *submenu-item* to the menubar, and places it in the submenu named by *submenu-title*. The new *submenu-item* description should be a list containing a string (or atom) and an optional function (e.g., `'("italic")` or `'("italic" italic-fn)`).

For example, the following lines will add new submenu-items to the DEMO-MENUBAR in Figure 3-5:

```
(garnet-gadgets:add-submenu-item DEMO-MENUBAR "face" '("outline"))
(garnet-gadgets:add-submenu-item DEMO-MENUBAR "size" '("very small")
                                          :before "small" :key #'car)
```

As shown in the second example, the *position* and *locator* parameters should correspond to existing submenu items.

3.10.2.3. Removing items from the menubar

Just as submenus and submenu-items can be added to the `menubar`, these two types of items can be removed.

`opal:Remove-Item` *menubar submenu* [*Method*]

This function removes the *submenu* from *menubar*. For traditional Garnet programming, the *submenu* should be a sublist of the top-level `:items` list, or just the title of a submenu (a string or atom).

For example, the following lines will remove an item from the DEMO-MENUBAR in Figure 3-5:

```
(opal:remove-item DEMO-MENUBAR
                  '("face" Face-Fn (("roman")("bold")("italic")("bold-italic"))))
(opal:remove-item DEMO-MENUBAR "size")
```

The following function is used to remove submenu-items from a `menubar`:

`gg:Remove-Submenu-Item` *menubar submenu-title submenu-item* [*Function*]

Submenu-item may either be the list description of the submenu-item (i.e., `("italic")`) or just the string (or atom) of the submenu-item (i.e., `"italic"`).

For example,

```
(garnet-gadgets:remove-submenu-item DEMO-MENUBAR "size" "small")
```

3.10.3. Programming the Menubar with Components

In the bottom-up approach to programming the `menubar`, the user must create components of the `menubar` (i.e., instances of `bar-item` and `submenu-item` gadgets) and attach them piece-by-piece. This design is loosely based on the interface to the Macintosh menubar in Macintosh Common Lisp. The functions for creating the components are described in section 3.10.3.2. Section 3.10.3.3 explains how to attach these components to the `menubar`.

3.10.3.1. An example

The code in Figure 3-6 creates a `menubar` and several component pieces, and then attaches the components to the `menubar`. This illustrates the bottom-up approach to programming the `menubar`.

Notice that the `menubar` instance must be added to the top-level aggregate before any bar-items are attached. This ensures that the `menubar` will be initialized with the proper main window before new submenu windows are added.

3.10.3.2. Creating components of the menubar

The functions in this section are used to create the three types of components that comprise a pull-down menu -- the `menubar` (the top-level part), the `bar-item` (which contains a submenu), and the `submenu-item`. Once the parts of the pull-down menu are created, they are attached using the functions of section 3.10.3.3. Please see section 3.10.3.3 for examples of the creation functions and attachment functions together.

`gg:Make-Menubar` [*Function*]

Returns an instance of `menubar`.

`gg:Make-Bar-Item &key` *desc font title* [*Function*]

This function returns an instance of `bar-item`. If any of the keys are supplied, then the corresponding slots of the `bar-item` instance are set with those values. The *desc* parameter is the description of a

```
(create-instance 'WIN inter:interactor-window
   (:left 750)(:top 80)(:width 200)(:height 200)
   (:title "Menubar Example"))
(s-value WIN :aggregate (create-instance 'TOP-AGG opal:aggregate))
(opal:update WIN)

; Create the menubar and the bar-item
(setf demo-menubar (garnet-gadgets:make-menubar))
(setf mac-bar (garnet-gadgets:make-bar-item :title "Mac Fonts"))

; Create submenu-items
(setf sub-1 (garnet-gadgets:make-submenu-item :desc '("Gothic")))
(setf sub-2 (garnet-gadgets:make-submenu-item :desc '("Venice")))
(setf sub-3 (garnet-gadgets:make-submenu-item :desc '("Old English")))

; Add the submenu-items to the bar-item
(opal:add-item mac-bar sub-1)
(opal:add-item mac-bar sub-2 :before sub-1)
(opal:add-item mac-bar sub-3 :after "Venice" :key #'car)

; Add the menubar to the top-level aggregate
(opal:add-component TOP-AGG demo-menubar)

; Add the bar-item to the menubar and update the window
(opal:add-item demo-menubar mac-bar)
(opal:update WIN)
```

Figure 3-6: The creation of a menubar and its components

submenu (e.g., `'("underline" NIL (("none")("single")("double")))`). The *font* is the font of the submenu-items within the submenu, and *title* is a string or atom that will be the heading of the submenu. If the title was already specified in the *desc* parameter, then no *title* parameter should be supplied.

`gg:Make-Submenu-Item &key` *desc enabled* [*Function*]

This function returns an instance of `submenu-item`. If any of the keys are supplied, then the corresponding slots of the `submenu-item` instance are set with those values. The *desc* parameter is the description of a submenu-item (e.g., `'("italic")` or `'("italic" italic-fn)`). The default for *enabled* is T.

3.10.3.3. Adding components to the menubar

Just as with the traditional Garnet approach, the two types of components that can be added to the `menubar` gadget are instances of the `bar-item` gadget and instances of the `submenu-item` gadget. The `add-item` method can be used to add new bar-items to a menubar, or to add new submenu-items to existing bar-items. Also, the following `Set-...` functions can be used to install a collection of components all at once.

`gg:Set-Menubar` *menubar new-bar-items* [*Function*]

Removes all current bar-items from *menubar* and installs the *new-bar-items*. The *new-bar-items* should be a list of `bar-item` instances.

`gg:Set-Submenu` *bar-item submenu-items* [*Function*]

Sets *bar-item* to have *submenu-items* in its submenu. *Submenu-items* is a list of `submenu-item` instances.

`opal:Add-Item` *menubar bar-item* `[[:where]` *position [locator]* `[:key` *index-function*`]]` [*Method*]
`opal:Add-Item` *bar-item submenu-item* `[[:where]` *position [locator]* `[:key` *index-function*`]]` [*Method*]

The *menubar*, *bar-item*, and *submenu-item* parameters above should be supplied with objects created by the functions in section 3.10.3.2. The optional parameters follow the standard `add-item` definition described in the Aggregadgets manual, and refer to the placement of the new `bar-item` among the existing bar-items. *Locator* may be either an existing `menubar` component, or some element of the `:items` list (like a submenu-title) used together with the *index-function* (see below).

After creating three `bar-item` instances, the example below shows how the bar-items can be attached as components to a `menubar`.

```
(setf bar1 (garnet-gadgets:make-bar-item
             :desc '("font-name" NIL (("courier") ("times") ("geneva")))))
(setf bar2 (garnet-gadgets:make-bar-item
             :desc '("other-fonts" NIL (("helvetica") ("chicago")))))
(setf bar3 (garnet-gadgets:make-bar-item
             :desc '("symbols" NIL (("mathematical") ("greek")))))
(opal:add-item DEMO-MENUBAR bar1)
(opal:add-item DEMO-MENUBAR bar2 :after "family" :key #'car)
(opal:add-item DEMO-MENUBAR bar3 :after bar2)
```

The following instructions show how submenu-items can be attached to oa `bar-item`. A `bar-item` object is first created, and then several submenu-items are attached to it using `add-item`:

```
(setf mac-bar (garnet-gadgets:make-bar-item :title "Mac Fonts"))
(setf sub-1 (garnet-gadgets:make-submenu-item :desc '("Gothic")))
(setf sub-2 (garnet-gadgets:make-submenu-item :desc '("Venice")))
(setf sub-3 (garnet-gadgets:make-submenu-item :desc '("Old English")))
(opal:add-item mac-bar sub-1)
(opal:add-item mac-bar sub-2 :before sub-1)
(opal:add-item mac-bar sub-3 :after "Venice" :key #'car)
```

3.10.3.4. Removing components from the menubar

The `bar-item` and `submenu-item` components can be removed from the `menubar` with the `remove-item` method:

`opal:Remove-Item` *menubar bar-item* [*Method*]

`opal:Remove-Item` *bar-item submenu-item* [*Method*]

For example, if we have already created a `bar-item` called BAR-1 and added it to DEMO-MENUBAR, then the following line will remove that item:

```
(opal:remove-item DEMO-MENUBAR bar1)
```

The `remove-item` method can also be used to remove submenu-items from bar-items. In order to remove a submenu item from the `bar-item` instance MAC-BAR, the following line can be used (provided SUB-1 is an existing `submenu-item` that was added to MAC-BAR):

```
(opal:remove-item mac-bar sub-1)
```

3.10.4. Finding Components of the Menubar

`gg:Menubar-Components` *menubar* [*Function*]

Returns a list of the `bar-item` instances installed in *menubar*.

`gg:Submenu-Components` *bar-item* [*Function*]

Returns a list of `submenu-item` instances that are installed in *bar-item*'s submenu.

`gg:Get-Bar-Component` *menubar item* [*Function*]

Returns the first `bar-item` object in *menubar* that corresponds to *item*. The *item* parameter may be a string or an atom, or one of the sublists of the *menubar*'s `:items` list.

`gg:Get-Submenu-Component` *bar-item item* [*Function*]

Returns the first `submenu-item` object in *bar-item* that corresponds to *item*. The *item* parameter may be a string or an atom, or a string/function pair that describes a `submenu-item`.

`gg:Find-Submenu-Component` *menubar submenu-title submenu-item* [*Function*]

Similar to `get-submenu-component`, except that `find-submenu-component` finds the appropriate `bar-item` instance in the given *menubar*. Returns the `submenu-item` object that corresponds to *submenu-item*. The parameter *submenu-title* should be the string or atom that is the title of some submenu. *Submenu-item* should be a string or atom, or a string/function pair that describes a `submenu-item` already installed in *submenu-title*.

3.10.5. Enabling and Disabling Components

`gg:Menubar-Disable-Component` *menubar-component* [*Function*]

Disables *menubar-component*'s interactors and makes its label grayed-out. The user will not be able to select *menubar-component* while it is disabled. *Menubar-component* is an instance of `bar-item` or `submenu-item`.

`gg:Menubar-Enable-Component` *menubar-component* [*Function*]

Enables *menubar-component*'s interactors and returns its label to solid text. *Menubar-component* is an instance of `bar-item` or `submenu-item`.

`gg:Menubar-Enabled-P` *menubar-component* [*Function*]

Returns T if the *menubar-component* is enabled. *Menubar-component* is an instance of `bar-item` or `submenu-item`.

3.10.6. Other Menubar Functions

`gg:Menubar-Get-Title` *menubar-component* [*Function*]

Returns the string or atom associated with *menubar-component*. The *menubar-component* must be an instance of a `bar-item` or `submenu-item` gadget.

`gg:Menubar-Set-Title` *menubar-component string* [*Function*]

Changes the title of *menubar-component* to *string* and, if *menubar-component* is installed in a `menubar`, sets the `:items` slot of the `menubar` appropriately. *Menubar-component* can be either an instance of `bar-item` or `submenu-item`. *String* is a string or an atom, suitable for putting in the `:items` slot. Returns *string*.

`gg:Menubar-Installed-P` *menubar-component* [*Function*]

Returns NIL if *menubar-component* is not attached to a `menubar`; otherwise returns the object it is installed in (either a `menubar` or a `bar-item`.

3.11. Labeled Box


```
(create-instance 'gg:Labeled-Box opal:aggregadget
   (:maybe-constant '(:left :top :label-offset :field-offset :min-frame-width
                      :label-string :field-font :label-font :visible))
   (:left 0)
   (:top 0)
   (:min-frame-width 10)
   (:label-offset 5)
   (:field-offset 6)
   (:label-string "Label:")
   (:value "Field")
   (:field-font opal:default-font)
   (:label-font (create-instance NIL opal:font (:face :bold)))
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

The loader file for the `labeled-box` is "labeled-box-loader".

The `labeled-box` gadget is comprised of a dynamic box with text both inside and to the left of the box. The text to the left of the box is a permanent label and may not be changed by the user. The text inside the box may be edited, however, and the width of the box will grow with the width of the string. As always, the current string inside the box may be accessed by the top level `:value` slot.

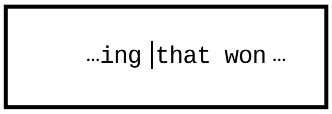
The width of the text frame will not fall below `:min-frame-width`.

The label to appear beside the box is in `:label-string`. The distance from the label to the left side of the box is specified in `:label-offset`, and the font of the label is in `:label-font`.

The distance from the box to the inner text is in `:field-offset`, and the font of the inner text is in `:field-font`.

3.12. Scrolling-Input-String

```
(create-instance 'gg:Scrolling-Input-String opal:aggregadget
   (:maybe-constant '(:left :top :width :font :line-style :visible))
   (:left 0)
   (:top 0)
   (:width 100) ; The width of the string area in pixels.
   (:value "Type here") ;The string that will originally appear in the
                        ;  box and that will be changed.
   (:selection-function NIL) ; Function to be executed after editing text
   (:font opal:default-font) ; **Must be fixed width**
   (:line-style opal:default-line-style)) ;line style can be used to set the color of the string
```



The loader file for the `scrolling-input-string` gadget is "scrolling-input-string-loader".

This allows the user to enter a one-line edited string of arbitrary length, but only requires a fixed area on the screen since if the string gets too long, it is automatically scrolled left and right as needed. Three little dots (an ellipsis) are displayed on either side of the string if any text is not visible on that side.

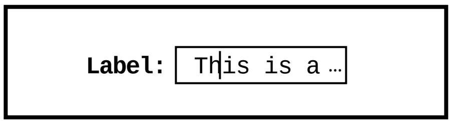
The user interface is as follows: To start editing, click with the left mouse button on the string. To stop, hit `return`. To abort, hit `^g`. If the string gets to be too large to fit into the specified width, then the string inside is scrolled left and right so the cursor is always visible. The cursor can be moved and text deleted with the usual editing commands (see the Interactors manual, page 170).

The top level `:value` slot is set with the final value of the string appearing inside the box. This slot may be set directly to change the initial value, and formulas may depend on it. A function may be specified in the `:selection-function` slot to be executed after the field text has changed (i.e., after the carriage return). Room is left on both sides of the string for a "..." symbol which shows whether the string has been scrolled or not. Therefore, the string will not appear exactly at the `:left` or extend the full `:width` (since room is left for the ...'s even when they are not needed).

3.13. Scrolling-Labeled-Box

```
(create-instance 'gg:Scrolling-Labeled-Box opal:aggregadget
   (:maybe-constant '(:left :top :width :label-offset :field-offset
                      :label-string :field-font :label-font :visible))
   (:left 0) (:top 0)
   (:width 130) ;The width of the entire area in pixels.   This must be big enough
                ;for the label and at least a few characters of the string!
   (:value "Field") ;The string that will originally appear in the
                    ;  box and that will be changed.
   (:selection-function NIL) ; Function to be executed after editing text
   (:field-font opal:default-font) ;**Must be fixed width**

   (:label-string "Label:") ; The string that will appear beside the box
   (:label-offset 5) ; The distance between the label and the box
   (:field-offset 2) ; The distance between the field text and the box
   (:label-font (create-instance NIL opal:default-font (:face :bold))))
                     ; The font of the string beside the box, can be variable-width
```



The loader file for the `scrolling-labeled-box` gadget is "scrolling-labeled-box-loader".

This is a combination of the `scrolling-input` gadget and the `labeled-box` gadget. It has a label and a box around the text. It operates just like the `scrolling-input-string`.

3.14. Graphics-Selection

```
(create-instance 'gg:Graphics-Selection opal:aggregadget
   (:start-where NIL)
   (:start-event :leftdown)
   (:running-where T)
   (:modify-function NIL)
   (:check-line T)
   (:movegrow-boxes-p T)
   (:movegrow-lines-p T)
   (:value NIL)
   (:active-p T)
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

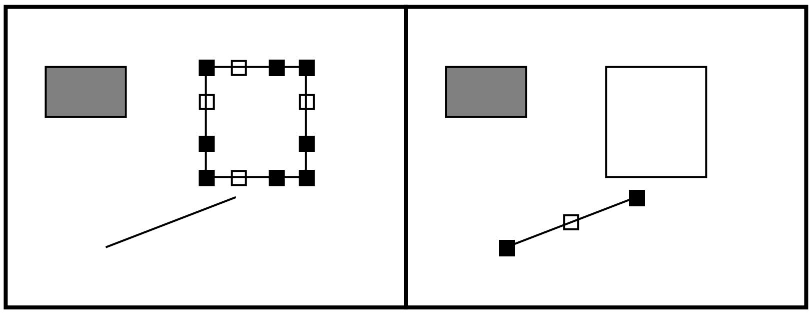


Figure 3-7: Selection of a rectangle and a line with `graphics-selection`

The loader file for `graphics-selection` is "graphics-loader".

The `graphics-selection` object is used to move and change the size of other graphical objects. (The `multi-graphics-selection` can select and move multiple objects -- see section 3.15.) When the user clicks on a graphical object (from a set of objects chosen by the designer), the object becomes selected and small selection squares appear around the perimeter of the object. The user can then drag the white squares to move the object or drag the black boxes to change the size of the object. Pressing in the background (i.e., on no object) causes the currently selected object to become unselected. Clicking on an object also causes the previously selected object to become unselected since only one object may be selected at a time. While moving and growing, if the mouse goes outside of `:running-where` or if the `^g` key is pressed, the operation aborts.

The `graphics-selection` gadget should be added as a component to some aggregate or aggregadget of the larger interface, just like any other gadget. The objects in the interface that will be affected by the `graphics-selection` gadget are determined by the slots `:start-where` and `:running-where`.

The `graphics-selection` gadget sets the `:box` slot of the object being moved or grown. This is consistent with the behavior of the `move-grow-interactor`, discussed in the Interactors manual. Therefore, you should create your objects with `:left`, `:top`, `:width`, and `:height` formulas that reference the `:box` slot.

The `:start-where` slot must be given a value to pass to an interactor to determine which items may be selected. The value must be one of the valid `...-or-none` forms for the interactors `:start-where` slot (see the Interactors Manual for a list of allowable values).

The `:start-event` slot specifies the event that will cause an object to be selected. The default is `:leftdown`, so if the left mouse button is clicked over an object in the `:start-where`, that object will become selected.

The `:running-where` slot is the area in which the objects can move and grow (see the Interactors Manual for allowable values).

If the `:check-line` slot is non-NIL, then the `graphics-selection` gadget will check the `:line-p` slot in the selected object, and if it is non-NIL then the interactor will select and change the object as a line. Instances of `opal:line` and `gg:arrow-line` already have their `:line-p` slots set to T. For other objects that should be selected as lines, the designer must set the `:line-p` slots explicitly (e.g., a composite object like an `arrow-line` is not really a line, though it should be treated like one).

If `:movegrow-lines-p` is NIL, then the `graphics-selection` object will not allow a user to drag the selection squares of a line, and a beep will be issued if the user clicks on a selection square of a line.

If `:movegrow-boxes-p` is NIL, then the `graphics-selection object` will not allow a user to drag the selection squares of a non-line, and a beep will be issued if the user clicks on a selection square of a non-line.

The `graphics-selection` gadget will be active when the value of its `:active-p` slot is T. To turn off the gadget, set this slot to NIL.

The `:selection-function` slot specifies a function to be executed upon the selection of any object by the user. This function must take the parameters:

```
(lambda (gadget-object new-selection))
```

The `new-selection` parameter may be NIL if no objects are selected (i.e., the user clicks in the background).

The designer can supply a `:modify-function` that will be called after an object is modified. It takes these parameters:

```
(lambda (gadget-object selected-object new-points))
```

The `new-points` will be a list of 4 numbers, either `left,top,width,height` or `x1,y1,x2,y2`.

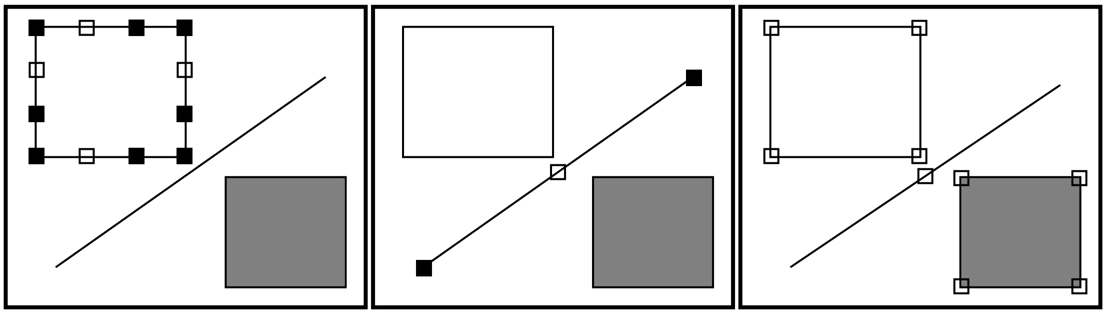
3.15. Multi-Graphics-Selection

```
(create-instance 'gg:Multi-Graphics-Selection opal:aggregadget
  ;; programmer-settable slots
   (:active-p T)        ;; whether objects can be selected with the gadget
   (:start-where NIL)  ;; supply a valid start-where here
   (:running-where T)  ;; if supplied, then this is the area in which the
                       ;; objects can move and grow
   (:selection-function NIL)  ;; this is called when the selection changes
   (:modify-function NIL)  ;; called when an object is changed size or position
   (:check-line T)          ;; whether to check for :line-p in objects
   (:check-polygon T)       ;; whether to check for :polygon-p in objects
   (:check-group T)         ;; whether to check for :group-p in objects
   (:check-grow-p NIL)      ;; whether to check for :grow-p in objects
   (:check-move-p NIL)      ;; whether to check for :move-p in objects
   (:move-multiple-p T)     ;; whether to move multiple objects as a group
   (:grow-multiple-p T)     ;; whether to grow multiple objects as a group
   (:input-filter NIL)      ;; used by the move-grow-interactor for gridding, etc.
   (:want-undo NIL)         ;; whether to save information to allow undo
   (:multiple-select-p T)  ;; if T, then multiple objects can be selected.

   (:other-multi-graphics-selection NIL]   ;; Used when several multi-selection gadgets in
                                           ;; different windows are working in conjunction.

   (:allow-component-select NIL)        ;; if T, then pressing with control will select
                                        ;; the component under the selected object.
   (:down-to-component-function gg::Point-To-Comp)  ;; a function that gets the
                                                    ;; appropriate component out
                                                    ;; of the object under the mouse.

  ;; slots the programmer can access
   (:current-selected-object NIL) ;; set with new selection or object to be moved
                                  ;; or grown before other slots are set.
   (:value NIL))   ;; current object or objects selected **DO NOT SET**
```



The loader file for `multi-graphics-selection` is "multi-selection-loader".

The `multi-selection` gadget is somewhat like the `graphics-selection`. The major difference is that multiple objects can be selected and manipulated by the user, and that the programmer must use a function to set the `:value` slot. Another difference is the way that the gadget checks whether move and grow is allowed.

This gadget exhibits the following features:

- Given a list of graphical objects, the `multi-graphics-selection` aggregadget will cause selection squares to appear on the bounding box of selected objects.
- One or more objects may be selected at a time, even when the objects are in different windows.
- A built-in interactor displays the selection squares around an object at the time of a specified event (such as clicking a mouse button on the object).
- Each selection square allows the user to move or grow the object by dragging the selection square.

- The user can move and grow several objects simultaneously.
- All of the objects inside a region (drawn by dragging the mouse) can be selected.

3.15.1. Programming Interface

Create an instance of the `gg:multi-graphics-selection` gadget and supply the `:start-where` slot with a valid list that can be passed to an interactor. This `:start-where` must return the items to be selected. It should be an `...-or-none` form, such as `:element-of-or-none`. An example of the parameter to `:start-where` is:

```
(list :element-of-or-none MYAGG)
```

The `:value` slot of the `multi-graphics-selection` object supplies the object(s) the user selects. If `:multiple-select-p` is NIL (the default), then it is a single object or NIL. If `:multiple-select-p` is T, then will always be a list or NIL (even if only one object is selected). Also, a `:selection-function` can be supplied and will be called each time the selection changes. It takes the parameters

```
(lambda (gadget new-selection)
```

where *new-selection* is the new value of `:value`.

When your interface contains selectable objects in several windows, you can put a multi-selection gadget in each window and coordinate them all. Each gadget's `:other-multi-graphics-selection` slot should contain a list of ALL the multi-selection gadgets. Then, each gadget's `:value` will reflect selections in all windows. A known bug is that the selection order is NOT preserved across multiple windows (you can't tell which object was selected first or last). Also, you cannot drag objects from one window to another.

The user can change the size and/or position of the objects by pressing on the selection handles (see below). If the `:check-line` slot is non-NIL, then the `:line-p` slot in the object returned by `:start-where` will be gvd, and if it is non-NIL then the interactor will change the object as a line. Note that instances of `opal:line` and `gg:arrow-line` have their `:line-p` slot set to T by default. For other objects, the programmer must set the `:line-p` slots explicitly. There is analogous interaction between the `:check-group` and `:check-polygon` slots of the gadget and the `:group-p` and `:polygon-p` slots of the selected objects.

The programmer can supply a `:modify-function` that will be called after an object is modified. It takes these parameters:

```
(gadget selected-object new-points)
```

The *new-points* will be a list of 4 numbers, either `left,top,width,height` or `x1,y1,x2,y2`.

Programmer-settable slots:

In summary, public slots of the `multi-graphics-selection` gadget are:

`:active-p` - If T, then the gadget will operate. If NIL, then none of the interactors will work. Setting to NIL does <u>not</u> clear the selection, however.

`:start-where` - Supply a valid start-where here.

`:running-where` - If supplied, then this is the area in which the objects can move and grow.

`:selection-function` - This is called when the selection changes.

`:modify-function` - This is called when an object is changed size or position.

`:check-line` - If T, the objects are checked for their `:line-p` slot and if that is non-NIL, then move or grown as a line.

`:check-polygon` - If T, the objects are checked for their `:polygon-p` slot and if that is non-NIL, then they are moved or grown as a polygon (by changing their `:point-list` slot).

`:check-group` - If T, the objects are checked for their `:group-p` slot and if that is non-NIL, then the individual components of the group are modified.

`:check-grow-p` - If T, then checks in each object to see if `:grow-p` is T, and if so, then will grow, else won't. If NIL, then everything can grow. Default NIL.

`:check-move-p` - If T, then checks in each object to see if `:move-p` is T, and if so, then will move, else won't. If NIL, then everything can move. Default NIL.

`:move-multiple-p` - If T, then if multiple items are selected and you press on a move box, then moves all of the objects. If NIL, then just moves the object you point to. Default=T.

`:grow-multiple-p` - If T, then when multiple items are selected, grow boxes appear at the corners of the whole selection, and pressing there will grow all the objects. If NIL, then those handles don't appear.

`:input-filter` - This is used by the move-grow-interactor for gridding. Consult the Interactors manual for a list of allowed values.

`:want-undo` - If T, then saves information (conses) so that you can call `undo-last-move-grow`.

`:allow-component-select` - Whether to allow selection of components (see below). Default=NIL.

`:down-to-component-function` - A function that determines which component of an object has just been selected (see below). Default=NIL.

`:multiple-select-p` - If T, then multiple objects can be selected. Default=NIL.

Slots that can be accessed:

`:value` - set with list of the current selections, in reverse order the user selected the objects (first selected object is last in the list). **Do not set this slot.** Instead, use the function `Set-Selection` (see below).

`:current-selected-object` - set with new selection before other slots are set.

Selecting components of the currently selected object:

You can enable the selecting of the components of the selected objects by setting `:allow-component-select` to T. For example, if the `:start-where` lists a set of objects, this feature can allow the selection of the *parts* of those objects. When component selection is enabled, then by pressing the `control-left` mouse button over a selected object, that object will be deselected, and its component will be selected instead. Similarly, if the `control-middle` mouse button or the `control-shift-left` mouse button is hit over a selected object, then that object is de-selected, and the object underneath is added to the selection set. The slot `:down-to-component-function` should contain a function to get the appropriate component out of the object under the mouse. This function might call a method in the selected object. Parameters are `(lambda obj x y)`. It should return the object to be selected, or NIL. The default function calls `opal:point-to-component` directly.

Slots of the objects that can be selected are:

`:line-p` - this should be T if the object should be moved as a line, and NIL if as a rectangle

`:group-p` - this should be T if the object is some instance of opal:aggregate and all its components should be moved as a group

`:polygon-p` - this should be T if the object is a polyline and it should be moved by changing its `:point-list` slot

`:points` - if `:line-p` is T, then the `:points` slot of the object is changed as the object is moved or grown.

`:box` - if `:line-p` is NIL, then the `:box` slot of the object is changed as the object is moved or grown.

`:grow-p` - if this object can be changed size

`:move-p` - if this object can be moved

Useful Functions:

`gg:Set-Selection` *gadget new-selection* [*Function*]

Gadget should be a `multi-graphics-selection` gadget, and *new-selection* is a list of objects that should be selected, or a single object to be selected, or NIL to turn off all selections. The list passed in is not damaged.

`gg:Undo-Last-Move-Grow` *multi-graphics-selection-gadget* [*Function*]

When `:want-undo` is non-NIL (default is NIL), then calling this function will undo the last move or grow and the selection will return to whatever it was when the objects were moved or grown. If you call `undo-last-move-grow` again, it undoes the undo (one-level undo). It is your responsibility to make sure that no objects were deleted or whatever between the grow and the call to undo.

Garnet does not yet have a general mechanism for Undo, so you should use this feature with care. It is currently your responsibility to keep track of what the last operation was and undo it.

3.15.2. End User Operation

The user can press on any object with the left button, and it will become selected. Pressing on the background, causes no object to be selected (the current object becomes de-selected). Selecting an object with the left button causes the previous object to be de-selected. If the application allows multiple selection, then clicking with shift-left or middle on an object toggles it in the selection set.

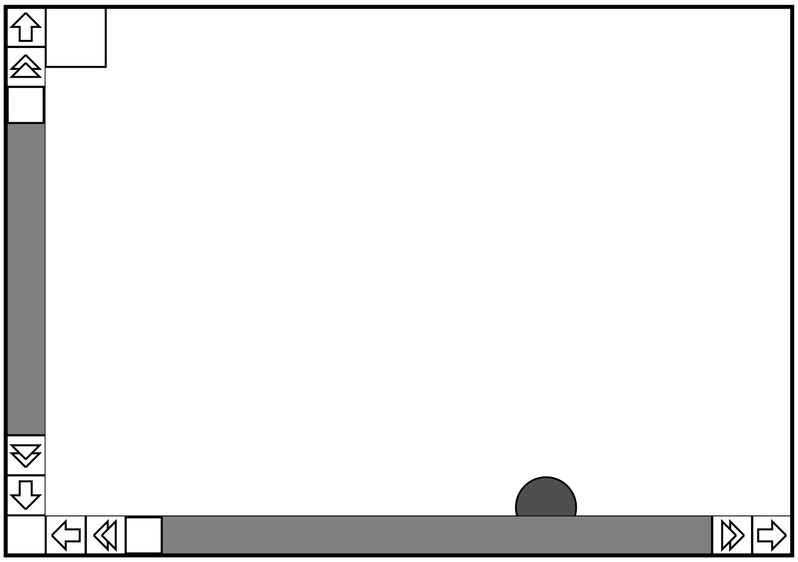
Once an object is selected, it can be grown by pressing with the left button on one of the black boxes or moved by pressing on a white box. While moving and growing, if the mouse goes outside of :running-where or if `^g` is pressed, the operation aborts.

The gadget also allows the user to change the size of several objects at once. When multiple objects are selected, outline handles appear around each object, and the whole set can be moved by pressing on any of these handles. Additionally, when `:grow-multiple-p` is non-NIL, black handles appear at the four corners of the collection of objects, and these can be used to scale the entire group.

The gadget also allows objects to be selected in a region. If you press down and drag before releasing, then only the objects fully inside the dragged rectangle will become selected. If you do this with the left button, then they will be selected. If you do this with shift-left or the middle button, then all objects inside the rectangle will be toggled in the selection set (added if not there, removed if there).

3.16. Scrolling-Windows

There are two scrolling-window gadgets which have the standard Garnet look and feel, and two other scrolling-window gadgets that have the Motif look and feel (see section 4.18). These windows are based on the design from Roderick J. Williams at the University of Leeds for the Garnet contest. The `scrolling-window` gadget allows you to do your own scrolling. The `scrolling-window-with-bars` gadget comes with a horizontal and vertical scroll bar, which you can have on either side (and can turn off explicitly). Each scroll bar will go blank if the entire area to be scrolled in is visible in the window.



```
(create-instance 'gg:Scrolling-Window opal:aggregadget
   (:maybe-constant '(:title :parent-window))
   (:left 0)  ; left, top, width and height of window
   (:top 0)
   (:position-by-hand NIL) ; if T, then left,top ignored and user asked for window position
   (:width 150) ;width and height of inside of outer window
   (:height 150)
   (:border-width 2) ; of window
   (:parent-window NIL) ; window this scrolling-window is inside of, or NIL if top level
   (:double-buffered-p NIL)
   (:omit-title-bar-p NIL)
   (:title "Scrolling-Window")
   (:icon-title (o-formula (gvl :title))) ;Default is the same as the title
   (:total-width 200)    ; total size of the scrollable area inside
   (:total-height 200)
   (:X-Offset 0)  ; offset in of the scrollable area
   (:Y-Offset 0)
   (:visible T)  ; whether the entire window is visible (mapped)

   ;; Read-Only slots
   (:Inner-Window NIL)  ; these are created by the update method
   (:inner-aggregate NIL) ; add your objects to this aggregate (but have to update first)
   (:outer-window NIL) ; call Opal:Update on this window (or on gadget itself)
```

```
(create-instance 'gg:Scrolling-Window-With-Bars opal:aggregadget
   (:maybe-constant '(:left :top :width :height :border-width :title
                     :total-width :total-height :h-scroll-bar-p :v-scroll-bar-p
                     :h-scroll-on-top-p :v-scroll-on-left-p :min-scroll-bar-width
                     :scr-trill-p :page-trill-p :indicator-text-p :h-scr-incr
                     :h-page-incr :v-scr-incr :v-page-incr :int-feedback-p
                     :indicator-font :parent-window :icon-title :visible))
   ;; Window slots
   (:left 0)  ; left, top, width and height of outermost window
   (:top 0)
   (:position-by-hand NIL) ; if T, then left,top ignored and user asked for window position
   (:width 150) ;width and height of inside of outer window
   (:height 150)
   (:border-width 2) ; of outer window
   (:parent-window NIL) ; window this scrolling-window is inside of, or NIL if top level
   (:double-buffered-p NIL)
   (:omit-title-bar-p NIL)
   (:title "Scrolling-Window")
   (:icon-title (o-formula (gvl :title))) ;Default is the same as the title
   (:total-width 200)    ; total size of the scrollable area inside
   (:total-height 200)
   (:X-Offset 0)  ; x offset in of the scrollable area. CANNOT BE A FORMULA
   (:Y-Offset 0)  ; CANNOT BE A FORMULA
   (:visible T)  ; whether the window and bars are visible (mapped)

   (:h-scroll-bar-p T)  ; Is there a horizontal scroll bar?
   (:v-scroll-bar-p T)  ; Is there a vertical scroll bar?

   ;; Scroll Bar slots
   (:h-scroll-on-top-p NIL)  ; whether horiz scroll bar is on top or bottom
   (:v-scroll-on-left-p T)   ; whether vert scroll bar is on left or right
   (:min-scroll-bar-width 20) ; these control both scroll bars
   (:scr-trill-p T)  ;single-line increment arrow buttons visible?
   (:page-trill-p T) ;page jump arrow buttons visible?
   (:h-scr-incr 10)  ; in pixels
   (:h-page-incr (o-formula (- (gvl :width) 10))) ; default jumps one page minus 10 pixels
   (:v-scr-incr 10)  ; in pixels
   (:v-page-incr (o-formula (- (gvl :height) 10))) ; default jumps one page minus 10 pixels
   (:int-feedback-p T) ; use NIL to have contents move continuously
   (:indicator-text-p NIL) ; Whether the pixel position is shown in the bars
   (:indicator-font (create-instance NIL opal:font (:size :small)))

   ;; Read-Only slots
   (:Inner-Window NIL)  ; these are created by the update method
   (:inner-aggregate NIL) ; add your objects to this aggregate (but have to update first)
   (:outer-window NIL) ; call Opal:Update on this window (or on gadget itself)
   (:clip-window NIL)
```

The loader file for both scrolling-window gadgets is "scrolling-window-loader".

Caveats:

- **If the scrolling-window has a** `:parent-window`**, update the parent window before instantiating the scrolling-window.**
- **Update the scrolling-window gadget before referring to its inner/outer windows and aggregates.**
- **The instance of the scrolling-window should <u>not</u> be added to an aggregate.**

These gadgets supply a scrollable region using the X window manager's ability to have subwindows bigger than the parent window. Garnet moves the subwindow around inside the parent window and X handles the clipping. All the objects in the window are instantiated (and therefore take up memory), but they will not be drawn if outside. You must specify the total area to be scrolled in using the `:total-width` and `:total-height` fields. (Therefore, the scrolling window gadgets do not support semi-infinite planes--you must pick a size for the user to scroll around in.) Often, you can compute the size based on the contents to be displayed in the window. There can be a formula in the `:total-*` fields, but it should have an initial value. *Note: It is illegal to have windows with a zero or negative width and* *height, so the* `:total-width` *and* `:total-height` *should always be greater than zero.*

The width and height specified for the window is the inside of the outer window, not counting the scroll bars. For `scrolling-windows`, this will usually be the same as the size of the visible region. For `Scrolling-Window-With-Bars`, the visible portion is smaller by the size of the scroll bars, which is usually the value of the `:min-scroll-bar-width` slot (unless you turn on indicator text).

Each of these gadgets is special in that they add themselves to the windows that they create. Since windows are not like other Gadgets, you need to follow special rules with scrolling windows.

First, *do not add scrolling-window or scrolling-window-with-bars gadgets to any aggregates or include them in aggregadgets*. If you want a scrolling window to be inside another window, you must use the `:parent-window` slot instead.

Second, *you must call* `opal:update` *on a scrolling window gadget immediately after creating it, and before adding anything to the windows.* The update method causes the windows to be created. If you want to create a prototype of a scrolling window (and specify special values for some of the fields), you can skip the update call, but then you cannot add any contents to the window.

The aggregate to add the contents to is provided in the slot `:inner-aggregate` of the gadget after the update call. To make the scrolling-window a subwindow of another window, specify the `:parent-window` of the scrolling-window. If you want to put a sub-window inside a scrolling-window, use the window in the `:inner-window` slot of the scrolling window as the `:parent` of the newly created window.

As an example:

```
(create-instance 'MYSCROLL garnet-gadgets:scrolling-window-with-bars
   (:left 650)(:top 10)(:width 300)(:height 400)
   ;;note that the next two formulas must have initial values
   (:total-width (o-formula (gvl :inner-aggregate :width) 200))
   (:total-height (o-formula (gvl :inner-aggregate :height) 200)))
(opal:update MYSCROLL) ;Must update scrolling windows before using them.
(opal:add-components (gv MYSCROLL :inner-aggregate)
               all the objects to be added to the scrolling window
               )
;;;create a scrolling window inside the other scrolling window, just for fun
(create-instance 'SUB-SCROLLING-WINDOW garnet-gadgets:scrolling-window-with-bars
   (:left 15)(:top 15)(:width 150)(:height 150)
   (:parent-window (gv MYSCROLL :inner-window)))
```

With `Scrolling-Windows`, but *not* `Scrolling-windows-with-Bars`, you can explicitly set the `:X-offset` and `:Y-Offset` fields using `s-value` to adjust the position of the contents. For `Scrolling-windows-with-Bars`, you must use the following procedures to have your application program scroll the window. This is necessary to get the scroll bars to be updated correctly to show the window position. These procedures also work with `Scrolling-Windows`.

Useful functions:

`gg:Scroll-Win-Inc` *scroll-win-gadget xinc yinc* [*Function*]

This function scrolls a window by adding the specified values, which can be negative. Note that *xinc* and *yinc* are usually zero or negative numbers, since they are the offset top-left corner of the inner window from the top-left of the clipping window, so to see down in the document, the inner window must be moved up.

`gg:Scroll-Win-To` *scroll-win-gadget x y* [*Function*]

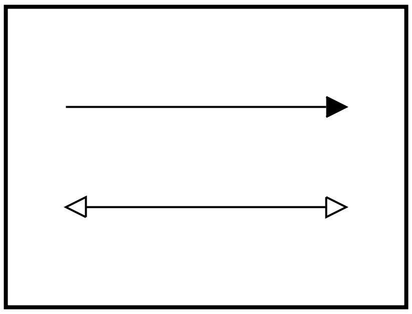
This function scrolls a window by putting the specified coordinate at the upper left corner of the clip window.

`gg:Show-Box` *scroll-win left top right bottom* [*Function*]

This function causes the scrolling-window *scroll-win* to scroll so that the region specified by *left*, *top*, *right* and *bottom* is visible. If the box is already visible, it will not cause the window to scroll. This can be used to cause the cursor in a text window, for example, or a "current item" to be visible. It is also used by the `focus-multifont` interactor.

If the box is larger than the visible region of the scrolling-window, the bottom and/or the rightmost parts of the box may remain hidden.

3.17. Arrow-line and Double-Arrow-Line



The `arrow-line` and `double-arrow-line` objects are comprised of a line and one or more arrowheads, effectively forming a single- or double-headed arrow. These objects are provided since the standard `opal:arrowhead` does not have an attached line.

3.17.1. Arrow-Line

```
(create-instance 'gg:Arrow-Line opal:aggregadget
   (:maybe-constant '(:x1 :y1 :x2 :y2 :line-style :open-p :filling-style :visible))
   (:X1 0) (:Y1 0)
   (:X2 20) (:Y2 20)
   (:line-style opal:default-line-style)
   (:filling-style NIL)
   (:open-p T))
```

The loader file for the `arrow-line` is "arrow-line-loader".

The origin (or tail) of the `arrow-line` is the point `(:x1,y1)`, and the tip is at `(:x2,y2)`. The values for these slots may be formulas that depend on the value of slots in other Garnet objects. For example, if `:x2` and `:y2` depended on the `:left` and `:top` coordinates of some rectangle, then the arrow would point to the top, left corner of the rectangle regardless of the movement of the rectangle.[5]

The appearance of the arrowheads themselves may also be customized. The `:line-style` slot contains a value indicating the thickness of all lines in the `arrow-line` object. Opal exports a set of pre-defined line styles, which must be preceded by the Opal package name, as in `opal:line-0`. Available line style classes are: `no-line`, `thin-line`, `line-0`, `line-1`, `line-2`, `line-4`, `line-8`, `dotted-line` and `dashed-line`. Other line style classes may also be defined (see the Opal Manual).

The slot `:filling-style` determines the shade of gray that will appear inside the arrowheads. Pre-defined filling styles are exported from Opal, and must again be preceded by the Opal package name. Available filling styles are `no-fill`, `black-fill`, `white-fill`, `gray-fill`, `light-gray-fill`, `dark-gray-fill`, and `diamond-fill`. The Opal function `halftone` may also be used to generate a filling style, as in `(:filling-style (opal:halftone 50))`, which is half-way between black and white fill.

[5]See the KR manual for a detailed discussion of constraints and formulas.

The slot `:open-p` determines whether a line is drawn across the base of the arrowhead.

Additional features of the arrowhead may be customized by accessing the slot `:arrowhead` of the `arrow-line`. For example, the following instruction would set the `:diameter` of an `arrow-line` arrowhead to 20:

```
(s-value (gv MY-ARROW-LINE :arrowhead) :diameter 20)
```

The same customization may also be implemented when the instance is created:

```
(create-instance 'MY-ARROW-LINE garnet-gadgets:arrow-line
   (:parts `(:line (:arrowhead :modify
                                (:diameter 20)))))
```

3.17.2. Double-Arrow-Line

```
(create-instance 'gg:Double-Arrow-Line opal:aggregadget
   (:maybe-constant '(:x1 :y1 :x2 :y2 :line-style :open-p :filling-style
                      :arrowhead-p :visible))
   (:X1 0) (:Y1 0)
   (:X2 40) (:Y2 40)
   (:line-style opal:default-line-style)
   (:filling-style NIL)
   (:open-p T)
   (:arrowhead-p :both))
```

The loader file for the `double-arrow-line` is "arrow-line-loader".

The endpoints of the `double-arrow-line` are at points `(:x1,:y1)` and `(:x2,:y2)`. The slots `:line-style`, `:filling-style`, and `:open-p` are used exactly as in the `arrow-line`, with both arrowheads taking identical properties.

The additional slot `:arrowhead-p` designates which end(s) of the line will have arrowheads. Allowed values are:

- `0` or `NIL` - No arrowheads
- `1` - Arrowhead at coordinate `(:x1,:y1)`
- `2` - Arrowhead at coordinate `(:x2,:y2)`
- `3` or `:both` - Arrowheads at both ends

The arrowheads may be further customized as in the `arrow-line` object. The arrowheads are available in the slots `:arrowhead1` and `:arrowhead2`.

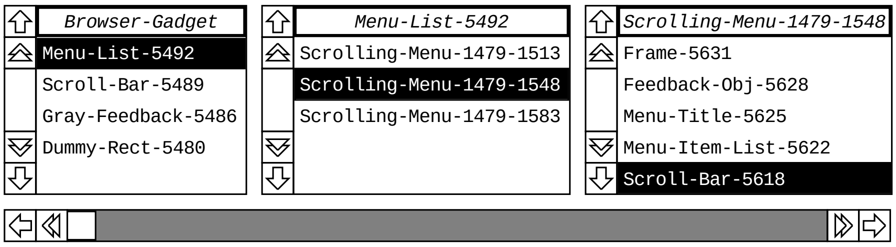
3.18. Browser Gadget

```
(create-instance 'gg:Browser-Gadget opal:aggregadget
   (:maybe-constant '(:left :top :num-menus :num-rows :menu-items-generating-function
                      :menu-function :item-to-string-function :additional-selection-p
                      :item-font :title-font :visible))

   ;; Browser parameters
   (:left 0)
   (:top 0)
   (:num-menus 3)
   (:num-rows 5)
   (:menu-items-generating-function NIL)
   (:item-to-string-function #'(lambda (item) item))   ;; assume item is a string

   ;; Additional-selection parameters
   (:additional-selection-p T)
   (:additional-selection (o-formula ... ))
   (:additional-selection-function NIL)
   (:additional-selection-coordinate NIL)

   ;; Scrolling-Menu parameters
   (:item-font opal:default-font)
   (:title-font (create-instance NIL opal:font (:face :italic)))
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```



The loader file for the `browser-gadget` is "browser-gadget-loader".

The `browser-gadget` is a sophisticated interface device which may be used to examine hierarchical structures, such as Garnet object networks and directory trees. The gadget is composed of a set of scrolling menus, where the selections in each scrolling menu correspond to the children of the item appearing in the title. Clicking on one of the menu selections causes that selection to appear as the title of the next scrolling menu, with all of its children appearing as choices in the new menu. Additionally, clicking the middle mouse button over a menu selection causes a gray feedback box to appear, indicating a secondary selection.

Two demos, named "demo-schema-browser" and "demo-file-browser", are included in the Garnet `demos` sub-directory as examples of how the `browser-gadget` is used in an interface. With the schema browser, the user may examine the inheritance and aggregate hierarchies of Garnet, while the file browser can be used to examine the file hierarchy of Unix directories.

3.18.1. User Interface

An instance of the `browser-gadget` may initially appear in a window with an item already displayed in the first menu. (Alternatively, the designer may provide a mechanism such as a `labeled-box` gadget through which the user initializes a fresh browser with the first item.) The selections in the first menu are derived from the item in the title through a specified function. When the user clicks the left mouse button on one of the menu choices, that selection will appear in the title of the next menu, with all of that item's "children" appearing as choices. If the item that the user selects does not generate any children, then a new menu is not generated.

The user may also click on a menu selection with the middle mouse button, causing the selection to be bordered by a gray rectangle. This selection is called the "additional selection", and there is only one for all of the menus in the `browser-gadget`.

The choices that are visible in each menu are controlled by the scroll bars appearing on the sides of the menus. If there are more menu selections derived from the title item than can be shown in a menu, then the background of the scroll bar will be gray and a white indicator box will appear. Clicking the left mouse button on the trill boxes at the top and bottom of the scroll bars will "scroll" more selections into the menu. Clicking on the single arrow trill boxes causes the visible selections to scroll one at a time, and clicking on the double arrow trill boxes will cause an entire "page" of new selections to appear (one page is equal to the number of items visible in the menu). The user may also drag the indicator of a scrolling menu scroll bar to adjust the visible selections.

Analogously, the horizontal scroll bar appearing below the menus may be adjusted to change which menus are displayed. When there are more menus to show than are allowed at one time, then the trill boxes can be clicked to scroll either one menu at a time or a whole "screen" of menus. Dragging the indicator in this scroll bar will cause a black rectangle to follow the mouse, rather than the indicator box itself. When the user releases the black rectangle, the indicator will jump to the position where it was released.

3.18.2. Programming Interface

3.18.2.1. Overview

It is important to note that the programming interface to the `browser-gadget` is different than in other Garnet gadgets. Due to the complexity of the gadget, this section is provided as a guide to the essential elements of the `browser-gadget` so that the designer can create and use an instance immediately. Subsequent sections describe in greater detail the slots and functions mentioned in this section.

When creating an instance of the `browser-gadget`, there is one slot that <u>must</u> be set. The slot `:menu-items-generating-function` must be provided with a function that generates children from the items that are to be shown in the titles of the menus. This function takes an item and returns a list of items that correspond to menu selections. These items can be of any type, but if they are not strings, then the slot `:item-to-string-function` must <u>also</u> be set with a function to derive strings from the items (its default value is the identity function). These functions are discussed further in section 3.18.3.

The `:items` slot adheres to the convention that if an element of this list is a list, then the second element is an item-function. The `:item-to-string-function` (described below) is applied to the first element of the item list to get a label for a menu selection. If data is to be stored in the elements of the `:items` list, it should be included as the third or greater elements in the item lists (see section 1.4.3).

To install an item in a `browser-gadget` instance, the function `set-first-item` should be called with the parameters of the name of the browser instance and the new item. A subsequent update of the window containing the instance will show the item appearing in the first menu with all of its children. Other functions used to manipulate the `browser-gadget` are discussed in section 3.18.6.

3.18.2.2. An example

Suppose that we want to define an instance of the `browser-gadget` to look at the inheritance hierarchy of Garnet schemas. First, create an instance called BROWSER-1 with the appropriate generating functions (these particular lambda-expressions are analyzed in 3.18.3).

```
(create-instance 'BROWSER-1 garnet-gadgets:browser-gadget
   (:menu-items-generating-function #'(lambda (item)
                                        (gv item :is-a-inv)))
   (:item-to-string-function #'(lambda (item)
                                  (if item
                                      (string-capitalize (kr:name-for-schema item))
                                      ""))))
```

The BROWSER-1 schema can be added to a Garnet window in the usual way:

```
(create-instance 'WIN inter:interactor-window
   (:width 600) (:height 200)
   (:aggregate (create-instance 'AGG opal:aggregate)))
(opal:add-component AGG BROWSER-1)
(opal:update WIN)
```

Now, we can initialize the BROWSER-1 object with a Garnet schema, such as the `opal:rectangle` schema:

```
(garnet-gadgets:set-first-item BROWSER-1 opal:rectangle)
(opal:update WIN)
```

All instances of `opal:rectangle` that currently exist will be shown in the first menu. Clicking on one of the selections in this menu will cause that selection to appear in the title of the second menu, with all of its instances as selections.

Since `opal:rectangle` is an instance of the `opal:graphical-object` schema, we can use the `push-first-item` (described in section 3.18.6) to show all of the objects that are instances of `opal:graphical-object`. If we call

```
(garnet-gadgets:push-first-item BROWSER-1 opal:graphical-object)
```

then the "Rectangle" title will be moved into the second menu along with all of its selections, and the "Graphical-Object" item will be displayed in the first menu with all of its instances. The "Rectangle" selection under the "Graphical-Object" title will be highlighted, since it was matched with the title of the second menu.

3.18.3. Generating Functions for Items and Strings

The slot `:menu-items-generating-function` contains a function which generates menu selections from each item in the scrolling menu titles. The function takes an *item* as a parameter, and returns a list of menu items which correspond to the selections in the scrolling menus. For example, if a `browser-gadget` instance is to be initialized with a Garnet schema, and the menus should display all of the instances of each item, then the `:menu-items-generating-function` appearing in the example of section 3.18.2.2 is appropriate. It should be noted that this function does not need to return a list of strings, but that eventually strings will be generated from the items that it returns (via the function in `:item-to-string-function`).

The function in the slot `:item-to-string-function` is used to generate strings from arbitrary items obtained from the `:menu-items-generating-function`. If the generated items are strings themselves, then the `:item-to-string-function` may retain its default value. The strings returned by the `:item-to-string-function` will be displayed as the titles and selections of the scrolling menus. In the example of section 3.18.2.2, the `:menu-items-generating-function` returns a list of Garnet schemas. So the supplied `:item-to-string-function` takes a schema as a parameter and returns the string name of the schema. Notice that when there are fewer items than there are menus, this function will generate empty strings for the titles of the blank menus.

3.18.4. Other Browser-Gadget Slots

The number of menus to be displayed horizontally in the `browser-gadget` is determined by the slot `:num-menus`. Since the set of menus in the gadget is implemented with an aggrelist, the menu objects will be adjusted automatically to correspond with the new value during the next call to `opal:update`. Analogously, the slot `:num-rows` determines the number of vertical selections to appear at one time in each scrolling menu.

The slots `:title-font` and `:item-font` control the fonts for the titles of the menus and the menu selections, respectively.

The function specified in `:selection-function` is executed when the user selects an item from one of the scrolling menus. The parameters of this function are

```
(lambda (browser-instance item))
```

where the *item* is an object generated by the function specified in `:menu-items-generating-function`. This function is executed after some internal bookkeeping is performed to update the `browser-gadget`.

3.18.5. The Additional Selection

When the user presses the middle mouse button over one of the scrolling menu selections, the outline of a gray rectangle will appear over the selection. The item chosen in this manner is called an "additional selection".

Whether this feature is active is determined by the value of the slot `:additional-selection-p`.

The item identified by the additional selection may be accessed through the slot `:additional-selection`. The value in this slot will correspond to some item returned by the function specified in `:menu-items-generating-function`. **Note:** this slot cannot be set directly to move the gray feedback box. Instead, the `:additional-selection-coordinate` slot must be set.

Since items may frequently be scrolled off to the side of the browser, it may not be possible to name explicitly the item which the gray feedback object should appear over. However, the "coordinate" of the additional selection can always be named in the slot `:additional-selection-coordinate`. This slot is set when the user selects the additional selection, and it may be set directly by the programmer. The `:additional-selection-coordinate` slot contains a list of two values -- the first is the rank of the menu which the selection appears in, and the second is the rank of the selection within the menu. Both ranks are zero-based, and are relative to the full lengths of the two item lists, not just the items currently visible.

The function specified in the slot `:additional-selection-function` will be executed when the user chooses the additional selection. The parameters are

```
(lambda (browser-instance item))
```

where *item* was just selected by the user. If the user presses over the previous additional selection, it will become deselected, and the `:additional-selection-function` will be called with NIL as the *item* parameter.

3.18.6. Manipulating the browser-gadget

Once an instance of the `browser-gadget` has been created, an item can be installed in the instance as starting object by calling `set-first-item` with the parameters

`gg:Set-First-Item` *browser-instance new-item* [*Function*]

The effect of calling this function is to install the *new-item* in the `:items` slot of the instance, and to initialize the bookkeeping slots of the instance.

The function `push-first-item` is used to add an item to the front of a `browser-gadget` instance. It takes the parameters

`gg:Push-First-Item` *browser-instance new-item* [*Function*]

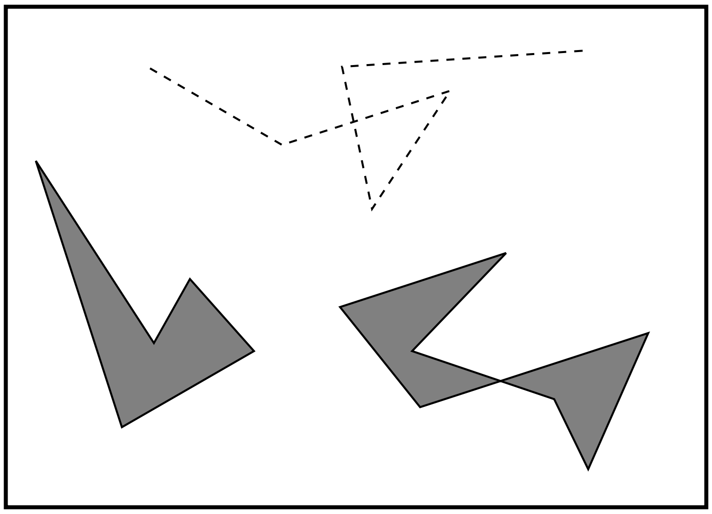
and adds the *new-item* to the front of the *browser-instance*'s `:items` list and adjusts the bookkeeping slots of the instance appropriately. A selection in the first menu is highlighted only if a match is found with the title of the second menu (which causes the browser to appear as though the second menu was actually generated from clicking on the selection in the first menu).

The function `promote-item` is used to install a new first item in an instance when the desired item already appears as a selection in one of the scrolling menus. The function is given the parameters

`gg:Promote-Item` *browser-instance coordinate* [*Function*]

where *coordinate* is a list of two numbers corresponding to the location of the desired item in the *browser-instance*. The syntax of the coordinate list is defined in section 3.18.5. If the item whose coordinate is passed is highlighted, then all of the menus to the right of the selection are retained; otherwise, the item becomes the only item in the instance.

3.19. Polyline-Creator



```
(create-instance 'gg:Polyline-Creator opal:aggregadget

    (:selection-function NIL) ; called when have full poly-line
    (:start-event :leftdown)   ; the event to start the whole process on
    (:start-where NIL)         ; where the mouse should be when the start-event happens
    (:running-where T)
    (:close-enough-value 3)    ; how close a point should be to the first point to stop the interaction
    (:input-filter NIL)

    ; Editing parameters
    (:mover-start-event :leftdown)      ; event to start moving a point
    (:mover-stop-event :leftup)         ; event to stop moving a point
    (:adder-start-event :leftdown)      ; event to add a point
    (:deleter-start-event :middledown)  ; event to delete a point
    (:threshold 3)                      ; how close to line to add a point
    (:polyline-being-edited NIL)        ; read-only slot

    ; Return value
    (:value NIL) ;set with final point list
```

The loader file for the `polyline-creator` gadget is "polyline-creator-loader". Examples of creating and editing polylines are in the GarnetDraw demo and the small `(gg:polyline-creator-demo-go)` which is loaded by default with the `polyline-creator`.

This gadget allows the user to enter new polylines (lists of points), while providing feedback. It also supports polyline editing, meaning that you can add, remove, and move points of a polyline with the mouse.

3.19.1. Creating New Polylines

The user interface for creating polylines is as follows: The user presses a button (specified in the `:start-event` slot) to start the interaction. Each subsequent button press causes a new segment to be added to the line. Feedback is provided to the user. The Polyline stops when:

- the new point is close enough (within `:close-enough-value` pixels) to the first point of the polyline (in which case the polyline is closed).
- a button pressed is different from the start event (in which case the polyline is open).
- the application calls the function `Stop-Polyline-Creator` (see below).

The gadget can also be aborted if the user types `^g` or the application calls `abort-polyline-creator`.

The function in the `:selection-function` is called to create the new polyline. This function should not destructively modify the point-list, but should instead *copy* the point-list if it will be changed. This functions is called with the parameters

```
(lambda (gadget new-point-list)
```

where *new-point-list* is of the form: `(x1 y1 x2 y2 x3 y3 ...)`.

The `:input-filter` slot is used just as in the `move-grow-interactor` and the `two-point-interactor`, described in the Interactors manual.

The `:value` slot is also set by the gadget with the final point-list. Applications are not allowed to set this directly (there can be no default value for this gadget).

3.19.2. Editing Existing Polylines

`gg:Toggle-Polyline-Handles` *polyline-creator-gadget polyline* [*Function*]

This function is used to display square "selection handles" on each point in the polyline to enable editing. The *polyline-creator-gadget* is passed as an argument to this function, since the selection handles to be displayed are components of the gadget.

To move a point, click the left mouse button over the point, move it to a new position, and release the left mouse button. Hitting `control-g` while moving a point will abort the move. Clicking the left mouse button in the middle of a line will add a point, after which the point can be dragged to a different location. Clicking on the background while editing a polyline will turn off the handles for the polyline.

There are several ways to delete points: either hit the middle mouse button over the point, double-click on the point, or hit the DELETE key while moving the point.

When the `toggle-polyline-handles` function is called, it first checks to see if the polyline is already being edited. If it is, it turns off the handles for the polyline. Otherwise, it turns on the handles for the polyline. Note that only one polyline can be edited at a time. If you call this function while a polyline is already being edited, it will turn off the handles for that polyline before turning on the handles for the polyline to be edited.

There are five slots in the polyline gadget which specify what actions cause editing. The slots and their default values are:

`:mover-start-event` - Default = `:leftdown`. The event to start moving a point.

`:mover-stop-event` - Default = `:leftup`. The event to stop moving a point.

`:adder-start-event` - Default = `:leftdown`. The event to add a point.

`:deleter-start-event` - Default = `:middledown`. The event to delete a point.

`:threshold` - Default = 3. How close you have to click next to a line to add a point.

There is a slot in the gadget called `:polyline-being-edited`. This slot will contain the polyline that is currently being edited, or NIL if no polyline is being edited.

3.19.3. Some Useful Functions

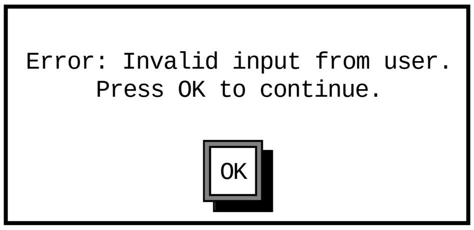
`gg:Stop-Polyline-Creator` *gadget* [*Function*]

This causes the gadget to create the current object. It ignores the current mouse position. This is useful if some other gadget (such as a palette changing the drawing mode) wants to stop the gadget. You can call this even if the gadget is not operating.

`gg:Abort-Polyline-Creator` *gadget* [*Function*]

This aborts the gadget without creating the polyline.

3.20. Error-Gadget




```
(create-instance 'gg:Error-Gadget opal:aggregadget
   (:parent-window NIL)
   (:font opal:default-font)
   (:justification :center)
   (:modal-p T)
   (:beep-p T)
   (:button-name "OK")
   (:window NIL)                ; Automatically initialized
   (:selection-function NIL)    ; (lambda (gadget value))
   ...)
```

The loader file for the `error-gadget` is "error-gadget-loader".

The `error-gadget` is a dialog box used to tell the user that an error has occurred. When activated, the user sees a window appear with a multi-line text message and an "OK" button centered in the window. If specified by the designer, all activities in the rest of the interface will be suspended until the user clicks on the "OK" button to cause the error window to disappear.

There is also a `motif-error-gadget`, which is described in section 4.12.

Some utility functions in section 3.20.2 allow you to easily raise an `error-gadget` in the context of checking user input for errors.

Caveats:

- **Update the parent window before instantiating the error-gadget.**
- **The instance of the error-gadget should not be added to an aggregate.**

3.20.1. Programming Interface

In order to associate an error window with an application, an instance of the `error-gadget` should be created with the `:parent-window` slot set to the window of the application. The error window is activated by calling one of the functions `gg:Display-Error` *error-gadget* `&optional` *message* [*Function*]

`gg:Display-Error-And-Wait` *error-gadget* `&optional` *message* [*Function*]

where the parameter *error-gadget* is the instance created by the user and *message* is a string to be displayed in the window. If *message* is not supplied, then the value in the `:string` slot of the gadget is used. The message may have multiple lines, indicated by carriage returns within the text string. While the `display-error` routine returns immediately when the dialog box appears, `display-error-and-wait` does not return until the user hits the OK button. The return value of both functions is always T.

When the error-gadget is associated with a parent window, the error window will appear centered inside of this window. If `:parent-window` is NIL, then the error window will appear at coordinates (200,200), relative to the upper left corner of the screen.

The font of the message is specified in the `:font` slot. The `:justification` slot is used to specify whether to align the text against the left or right margin of the window or whether each line should be centered in the window (allowed values are `:left`, `:right`, and `:center`).

If the value of the `:modal-p` slot is T, then all interactors in the rest of the interface will be suspended, and the user will not be able to continue working until the "OK" button has been pressed. If `:modal-p` is NIL, then the interface will continue to function with the error window visible.

If the `:beep-p` slot is T, then Garnet will sound a beep when the gadget becomes visible. To turn off the beep, set `:beep-p` to NIL.

The `:button-name` slot determines the label of the button. Since the `display-error` routines do *not* take this as a parameter, it must be set in the gadget itself.

After the instance of the `error-gadget` has been created, the window which will contain the text and the button may be accessed through the `:window` slot of the instance. Note: When the `error-gadget` instance has a parent-window, the `:left` and `:top` coordinates of this window will be relative to the parent-window. Otherwise, they are relative to the full screen.

3.20.2. Error-Checking and Careful Evaluation

There are several functions that can be used to evaluate lisp expressions that may contain errors, while avoiding a crash into the debugger. These functions may be used to evaluate user input to make sure it is free of errors before passing it on to the rest of an application. If the user input contains an error (i.e., does not successfully evaluate), the functions return a special value and can display an `error-gadget` informing the user of the error.

These functions are more portable and more useful than implementation-dependent functions like `ignore-errors`. These functions are used in many Garnet applications and demos where information is supplied by the user. Examples can be found in the `Inspector`, `demo-graph`, `garnet-calculator`, and the line and filling-style dialog boxes in Gilt.

All of the `careful-eval` functions are defined in `error-gadget-utils`, and are loaded automatically along with the error and query gadgets when you do `(garnet-load "gadgets:error-gadget-loader")` or `(garnet-load "gadgets:motif-error-gadget-loader")`.

These functions were inspired by the `protected-eval` module in the Garnet `contrib` directory, created by Russell G. Almond.

3.20.2.1. Careful-Eval

`gg:Careful-Eval` *form* `&optional` *error-gadget error-message* [*Macro*]

`Careful-Eval` will evaluate the *form*. If an error is encountered during the eval, then the *error-gadget* will be displayed with the actual lisp error message that was generated, followed by the specified *error-message* (separated by carriage returns).

When the evaluation is successful, `gg:Careful-Eval` returns the evaluated value (which may be multiple values). If there was an error, then `Careful-Eval` returns two values: NIL and the error condition structure. (For a discussion of error conditions, see Chapter 29 of the Second Edition of Guy Steele's *Common Lisp, the Language*.)

Examples:

```
lisp> (gg:careful-eval '(+ 4 5))     ;; evaluates successfully
9
lisp> (gg:careful-eval '(+ 4 y))     ;; signals an error
NIL
#<EXCL::SIMPLE-ERROR.0>
lisp> (multiple-value-bind (val errorp)
          (gg:careful-eval '(+ 4 y))
()        (if errorp   ; perhaps (typep errorp 'condition) is safer
            (format t "An error was encountered~%")
            (format t "Value is ~S~%" val)))
An error was encountered

NIL
lisp>
```

3.20.2.2. Careful-Read-From-String

`gg:Careful-Read-From-String` *string* `&optional` *error-gadget error-message* [*Function*]

`Careful-Read-From-String` will try to read a symbol or expression from the *string* and return it if successful. If an error is encountered, then the *error-gadget* will be raised and two values will be returned: NIL and the error condition. The message displayed in the error gadget will be a concatenation of the actual lisp error message followed by the *error-message*.

3.20.2.3. Careful-String-Eval

`gg:Careful-String-Eval` *string* `&optional` *error-gadget error-message* [*Function*]

`Careful-String-Eval` will try to read a symbol or expression from the string and then eval it. If the read and eval are successful, then the evaluated value is returned. If there was an error during either the read or eval, then the *error-gadget* is raised and two values are returned: NIL and the error condition. The message displayed in the error gadget will be a concatenation of the actual lisp error message and the *error-message*.

3.20.2.4. Careful-Eval-Formula-Lambda

`gg:Careful-Eval-Formula-Lambda` *expr error-gadget error-message*
the-obj the-slot the-formula warn-p [*Function*]

`Careful-Eval-Formula-Lambda` evaluates the expression AS IF it were installed in *the-slot* of *the-obj* as a formula. This is useful when the *expr* contains `gvl` calls, which normally require that the *expr* is already installed in an `o-formula` when it is evaluated. If the evaluation is successful, then the evaluated value is returned. If there was an error during the eval, then the *error-gadget* is raised and two values are returned: NIL and the error condition. The message displayed in the error gadget will be a concatenation of the actual lisp error message followed by the *error-message*.

If a formula object has already been created for the expression, then it should be passed as the value of *the-formula*. This will cause dependencies to be established as the `gv`'s and `gvl`'s are evaluated in the expression. *The-formula* may also have the value `:ignore`, which will prevent the establishment of dependencies.

Example:

```
lisp> (create-instance 'R opal:rectangle
        (:my-left 67))
Object R
#k<R>
lisp> (gg:careful-eval-formula-lambda '(gvl :my-left) NIL NIL
                                        R :left :ignore NIL)
67
lisp>
```

3.21. Query-Gadget

```
(create-instance 'gg:Query-Gadget gg:error-gadget
   (:button-names '("OK" "CANCEL"))
   (:string "Is that OK?")
   (:parent-window NIL)
   (:font opal:default-font)
   (:justification :center)
   (:modal-p T)
   (:beep-p T)
   (:window NIL)               ; Automatically initialized
   (:selection-function NIL)   ; (lambda (gadget value))
   ...)
```

The loader file for the `query-gadget` is "error-gadget-loader" (the `query-gadget` is in the same file as the `error-gadget`).

The `query-gadget` is similar to the `error-gadget`, but it allows more buttons in the window, so it is useful for a general purpose dialog box. The button names are supplied in the `:button-names` slot of the `query-gadget` or as a parameter to the display functions. The use of the `query-gadget` is the same as the `error-gadget` (and the same caveats apply). There is also a `motif-query-gadget`, which is described in section 4.13.

To display a query-gadget, you first create an instance of `query-gadget`, and then call one of:

`display-query` *query-gadget* `&optional` *message label-list*

`display-query-and-wait` *query-gadget* `&optional` *message label-list*

The *message* is the string to display, and the optional *label-list* allows you to change the buttons. It should be a list of strings, atoms or keywords. If *message* is not supplied, then the value of the `:string` slot of the gadget is used. This function displays the query-gadget on the screen and then returns immediately. The `selection-function` of the query gadget (if any) is called with the item from the label-list the user selected. While the `display-query` routine returns immediately when the dialog box appears, `display-query-and-wait` does not return until the user hits one of the buttons. The return value `display-query-and-wait` is the label of the selected button.

3.22. Save Gadget

```
(create-instance 'gg:Save-Gadget opal:aggregadget
  (:maybe-constant '(:parent-window :window-title :window-left :window-top
                     :message-string :num-visible :initial-directory :button-panel-items
                     :button-panel-h-spacing :min-gadget-width :modal-p
                     :check-filenames-p :query-message :query-buttons
                     :dir-input-field-font :dir-input-label-font :message-font
                     :file-menu-font :file-input-field-font :file-input-label-font
                     :button-panel-font))
  (:parent-window NIL)
  (:window-title "save window")
  (:min-gadget-width 240)
  (:initial-directory "./")
  (:message-string "fetching directory...")
  (:query-message "save over existing file")
  (:button-panel-items '("save" "cancel"))
  (:button-panel-h-spacing 25)
  (:num-visible 6)
  (:check-filenames-p t)
  (:modal-p NIL)
  (:selection-function NIL)    ; (lambda (gadget value))

  (:dir-input-field-font (opal:get-standard-font NIL NIL :small))
  (:dir-input-label-font (opal:get-standard-font NIL :bold NIL))
  (:file-input-field-font (opal:get-standard-font NIL NIL :small))
  (:file-input-label-font (opal:get-standard-font NIL :bold NIL))
  (:message-font (opal:get-standard-font :fixed :italic :small))
  (:button-panel-font opal:default-font)
  (:file-menu-font (opal:get-standard-font NIL :bold NIL))
  ...)
```

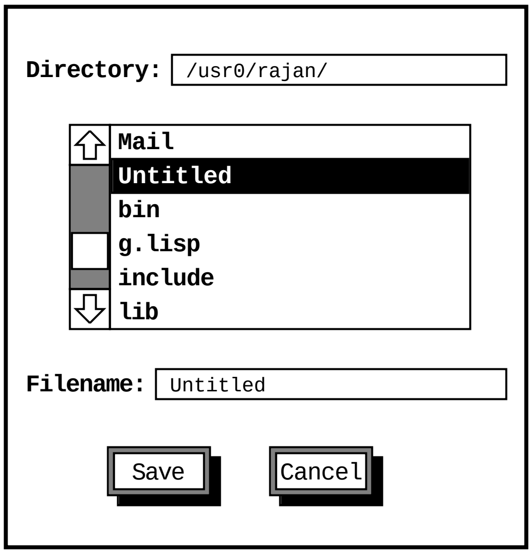


Figure 3-8: A save-gadget showing the contents of directory `/usr0/rajan/`

The loader file for the `save-gadget` is "save-gadget-loader" (which also loads the `load-gadget`). Figure 3-8 shows a picture of the save gadget.

The `save-gadget` is a dialog box used to save a file, while displaying the contents of the destination directory in a scrolling menu. The gadget has an accompanying query-gadget dialog box (not shown) that can ask the user if the file really should be saved before the `save-gadget` appears. This is an extra level of convenience for the application designer.

There is also a `motif-save-gadget`, as well as a `load-gadget` and `motif-load-gadget`.

Caveats:

- **Update the parent window before instantiating the save-gadget.**
- **The instance of the save-gadget should not be added to an aggregate.**

3.22.1. Programming Interface

When a save gadget is created, it does not appear automatically. Like the query and error gadgets, it has its own display function. The save window is activated by calling one of these functions:

`gg:Display-Save-Gadget` *save-gadget* `&optional` *initial-filename* [*Function*]

`gg:Display-Save-Gadget-And-Wait` *save-gadget* `&optional` *initial-filename* [*Function*]

While the `display-save-gadget` routine returns immediately when the dialog box appears, `display-save-gadget-and-wait` does not return until the user hits either the "Save" or "Cancel" button. If an *initial-filename* is provided, it will appear in the "Filename:" box when the gadget is displayed.

NOTE: To change the directory, set the `:initial-directory` slot of the gadget to be the new directory. Then, when you call one of the display methods, the directory will be updated.

To hide a save window, use

`gg:Hide-Save-Gadget` *save-gadget* [*Function*]

The following function is described in section 3.22.4.

`gg:Save-File-If-Wanted` *save-gadget* `&optional` *filename* (*query-string* `"Save file first?"`) [*Function*]

When a save-gadget is first displayed, the "Directory" box will contain the present directory (unless otherwise specified, as explained in the next section); the scrolling-menu will have the contents of that directory; and the "Filename" box will be blank.

Whenever the directory name is changed by the user, the scrolling menu will also change to list the contents of the new directory. If an invalid directory name is specified, there will be a beep and the invalid name will be replaced by the previous name. Whenever a directory is being fetched, a brief message (by default, "Fetching directory...") will appear, and will go away when the scrolling menu has been updated. When a file name is typed into the "Directory" box, the file name will be moved down to the "Filename" box, and the menu will be updated.

If a file in the scrolling menu is selected, then the "Filename" box will contain the name of that file. If a directory is selected, the "Directory" box will be set to the selected directory, and the scrolling menu will once again update itself.

If an invalid file name is typed into the "Filename" box, there will be a beep and the "Filename" box will be reset. An invalid file name is one that has a directory name in it ("/usr/garnet/foo", for example).

The following slots may be changed to customize the `save-gadget`:

`:window-title` - contains the title of the save window, which is by default "Save Window". Window managers usually do not display titles for subwindows (i.e., if a window is specified in `:parent-window`).

`:parent-window` - if this slot contains a window, then the `save-gadget` will appear as a subwindow of that window. By default, the gadget will automatically be centered inside the parent window. If this is not desired, the `:window-left` and `:window-top` slots can be changed to position the gadget.

`:window-left` and `:window-top` - specify the coordinates of the dialog box. Default values are 0 for both slots unless there is a parent window.

`:initial-directory` - the directory to display when the `save-gadget` appears. The default is "./", which is the current directory as determined by the lisp process.

`:message-string` - the message to display to the user while the save gadget fetches the contents of a new directory. Default is "Fetching directory...".

`:num-visible` - how many files to display in the scrolling menu. Default is 6.

`:button-panel-items` - a list of names for the buttons. The default is '("Save" "Cancel"). NOTE: It is important that, when you rename the buttons and use the `default-save-function`, you rename them in the "Save" "Cancel" order. That is, the label that should cause the gadget to save must appear first in the `:items` list, and the label that cancels the gadget's action must appear second. For example, if you rename the `:button-panel-items` slot as '("Go" "Return"), it will produce the correct results. However, if you use '("Return" "Go") instead, the wrong functions will get called.

`:button-panel-h-spacing` - the distance between the buttons (default 25).

`:min-gadget-width` specifies the width of the "Directory" and "Filename" boxes. The scrolling menu is centered between them.

`:modal-p` - when T, then interaction in other Garnet windows will be suspended untill either the "Save" or the "Cancel" button is hit.

`:check-filenames-p` - whether to check to see whether the file already exists before saving. If the file exists, then a query gadget will pop up and ask for confirmation.

`:query-message` - the string that will be used in the query gadget that pops up when you try to overwrite a file. If `:check-filenames-p` slot is NIL, this slot is ignored.

`:selection-function` - as usual, the function called when the "Save" button is hit.

`:dir-input-field-font` and `:dir-input-label-font` - the fonts for the field and label of the "Directory" box.

`:file-input-field-font` and `:file-input-label-font` - the fonts for the field and label of the "Filename" box.

`:message-font` - the font to use for the message that appears when the directory is being fetched.

`:file-menu-font` - the font of the items inside the scrolling menu

`:button-panel-font` - the font for the buttons

3.22.2. Adding more gadgets to the save gadget

It is possible to add more gadgets, such as extra buttons, etc. to the save gadget. To do this, you simply add more components to the `:parts` list of the save gadget (which is an aggregadget). However, you MUST include the following 5 components in the parts list: `:dir-input`, `:file-menu`, `:file-input`, `:message`, and `:OK-cancel-buttons`.

An example of adding more gadgets to a save gadget follows:

```
(create-instance 'SG gg:save-gadget
  (:parts
   `(:dir-input :file-menu :file-input :message :OK-cancel-buttons
     (:extra-button ,gg:text-button
      (:left 10) (:top 220)
      (:text-offset 2) (:shadow-offset 5) (:gray-width 3)
      (:string "Test")))))
```

This will, in addition to creating the standard save gadget parts, create an additional button. This button can be accessed by using `(gv SG :extra-button)`. Naturally, you can have selection functions, etc. to whatever gadgets you add. However, it is extremely important to include the `:dir-input`,

`:file-menu`, `:file-input`, `:message` and `:OK-cancel-buttons` in the `:parts` list.

NOTE: The save/cancel buttons automatically position themselves 25 pixels below the last gadget in the `:parts` list, since most people desire the buttons at the bottom of the gadget. If this is not desired, you can modify the `:top` slot of the `:OK-cancel-buttons`.

3.22.3. Hacking the Save Gadget

The slots described above should be enough to customize most applications. However, when that is not the case, it is possible to hack the save gadget.

For example, the save/cancel buttons are centered with respect to the "Filename" box. If this is not desirable, the `:OK-cancel-buttons` slot can be modified to the desired left and top coordinates.

Suppose the left of the save/cancel buttons should be at 10. The save gadget then would look like:

```
(create-instance ’sg gg:save-gadget
  (:parts
    ‘(:dir-input
      :message
      :file-menu
      :file-input
      (:OK-cancel-buttons :modify (:left 10)))))
```

3.22.4. The Save-File-If-Wanted function

If you are using a menubar with a "File" menu, you might want to use the `save-file-if-wanted` function. You would call this function before such operations as quit, close, and read if the contents of the window had not yet been saved. The format for this function is:

`gg:Save-File-If-Wanted` *save-gadget* `&optional` *filename* `(`*query-string* `"Save file first?")` [*Function*]

This function will pop up a query gadget that asks "Save file first?", or whatever you specify as the *query-string*. If "Yes" is selected, then it will call the standard `display-save-gadget-and-wait` function on the given filename, and the return value of this function will be the same as the return value for the *save-gadget*’s `:selection-function`. If "Cancel" is selected, it will return `:CANCEL`. If "No" is selected, it will return `:NO`.

For an example of when and where this function can be used, look at the source code for Garnetdraw, under the section labeled "MENU FUNCTIONS AND MENUBAR". The Open, New and Quit functions all call this function.

Often, it is necessary to know if the "Cancel" button was hit or not. For this purpose, the functions `save-file-if-wanted` and the `display-save-gadget-and-wait` return `:cancel` if the "Cancel" button was hit. For example, the quit function in Garnetdraw looks like this:

```
(defun quit-fun (gadget menu-item submenu-item)
  (unless (eq :cancel (gg:Save-File-If-Wanted *save-db* *document-name*))
    (do-stop)))
```

If the user clicks on "Cancel" either in the "Save file first?" query box, or in the save-gadget itself, `save-file-if-wanted` will return `:cancel`.

3.23. Load Gadget

```
(create-instance 'gg:Load-Gadget opal:aggregadget
  (:maybe-constant '(:parent-window :window-title :window-left :window-top
                     :message-string :num-visible :initial-directory :button-panel-items
                     :button-panel-h-spacing :min-gadget-width :modal-p
                     :check-filenames-p :dir-input-field-font :dir-input-label-font
                     :message-font :file-menu-font :file-input-field-font
                     :file-input-label-font :button-panel-font))
  (:parent-window NIL)
  (:window-title "load window")
  (:min-gadget-width 240)
  (:initial-directory "./")
  (:message-string "fetching directory...")
  (:button-panel-items '("load" "cancel"))
  (:button-panel-h-spacing 25)
  (:num-visible 6)
  (:check-filenames-p t)
  (:modal-p nil)
  (:selection-function NIL)    ; (lambda (gadget value))

  (:dir-input-field-font (opal:get-standard-font nil nil :small))
  (:dir-input-label-font (opal:get-standard-font nil :bold nil))
  (:file-input-field-font (opal:get-standard-font nil nil :small))
  (:file-input-label-font (opal:get-standard-font nil :bold nil))
  (:message-font (opal:get-standard-font :fixed :italic :small))
  (:button-panel-font opal:default-font)
  (:file-menu-font (opal:get-standard-font nil :bold nil))
  ...)
```

The `load-gadget` is loaded along with the `save-gadget` by the file "save-gadget-loader".

The `load-gadget` is very similar to the `save-gadget`. Both look alike, except for their window titles. The same caveats apply to both the save and load gadgets (see section 3.22).

The `load-gadget` has its own functions for displaying and hiding the gadget, which are analogous to those used by the `save-gadget`:

`gg:Display-Load-Gadget` *load-gadget* `&optional` *initial-filename* [*Function*]

`gg:Display-Load-Gadget-And-Wait` *load-gadget* `&optional` *initial-filename* [*Function*]

`gg:Hide-Load-Gadget` *load-gadget* [*Function*]

When a load gadget is created and `display-load-gadget` is called, the window that pops up contains the same initial contents as in the save gadget. The "Directory" box, the scrolling-menu, and the message, all work identically in both the gadgets.

The "Filename" box resembles the save gadget in that it beeps when an invalid file name is typed in (unless the `:check-filenames-p` slot is NIL), and is reset to the empty string, "". However, an invalid file name is defined as a file name that does not exist, or a directory.

As in the save gadget, when you rename the buttons and use the default load function, it is important to put the name corresponding to the "Load" button as the first element of the `:button-panel-items` list.

3.24. Property Sheets

The `prop-sheet` gadget takes a list of values to display, and `prop-sheet-for-obj` takes a KR object to display. The `prop-sheet-with-OK` and `prop-sheet-for-obj-with-OK` gadgets combine a property sheet with OK, Apply and Cancel buttons and functions to display these in windows (using the Garnet look and feel). Similarly, the `motif-prop-sheet-with-OK` and `motif-prop-sheet-for-obj-with-OK` combine a property sheet with buttons, but use the Motif look and feel (see section 4.16).

3.24.1. User Interface

Press on the value of a slot with the left button to begin typing. Press with the left button again (anywhere) or hit `return` or `^j` to stop editing (if multi-line strings are allowed, then `return` goes to the next line, so you need to use `^j` or left button to stop editing). Pressing with any other button inside the string moves the cursor. Regular editing operations are supported (see the text-interactor in the Interactors manual). If you hit `tab`, the cursor will move to the next field. If label selection is enabled, then labels can be selected by pressing with any mouse button. If value selection is enabled, then values must be selected with the *right* button while they are not being edited. Selected labels or values are displayed in bold.

3.24.2. Prop-Sheet

```
(create-instance 'gg:Prop-Sheet opal:aggregadget
    (:maybe-constant '(:left :top :items :default-filter :v-spacing
                       :multi-line-p :select-label-p :visible
                       :label-selected-func :label-select-event
                       :select-value-p :value-selected-func :single-select-p))
   ; Customizable slots
    (:left 0) (:top 0)
    (:items NIL) ; put the values to be displayed here
    (:default-filter 'default-filter)
    (:v-spacing 1)
    (:pixel-margin NIL)
    (:rank-margin NIL)
    (:multi-line-p NIL) ; T if multi-line strings are allowed
    (:select-label-p NIL) ; T if want to be able to select the labels
    (:label-selected-func NIL)
    (:label-select-event :any-mousedown)
    (:select-value-p NIL) ; if want to be able to select the values
    (:value-selected-func NIL)
    (:single-select-p NIL) ; to select more than one value or label

   ; Read-only slots
    (:label-selected NIL) ; set with the selected label objects (or a list)
    (:value-selected NIL) ; set with the selected value objects (or a list)
    (:value ...)   ; list of pairs of all the slots and their (filtered) values
    (:changed-values NIL)) ; only the values that have changed
```

```
COLOR: Red
HEIGHT: 34
STATUS: Nervous
DIRECTION: up down diagonal
RANGE (1..20): 1
```

Figure 3-9: Example of a property sheet with an embedded gadget.

The loader for the `gg:prop-sheet` gadget is "prop-sheet-loader".

Customizable slots:

`:left`, `:top` - Position of the gadget. Default: 0,0

`:items` - The control list of the items to be displayed in the gadget. The format for the list is a list of lists, as follows:

```
( (label1 stringval1 [filter1 [realval1 [comment]]]) (label2 ...) )
```

- The `labels` can be atoms or strings, and are shown at the left.

- The `stringval` is the initial (default) value displayed. For an example of the use of the various forms of `stringval`, see section 3.24.9. It can be:
  - a string,
  - a formula object which computes a string. Note that all references in the formula must be absolute (since otherwise they would be relative to the property sheet).
  - an instance of a gadget (e.g., a `radio-button-panel`), in which case that instance is used instead of an editable text field. Note that the instance itself is used, so it will be destroyed if the `prop-sheet` is destroyed. The gadget instance should supply its value in a slot called `:value` (as the standard garnet gadgets do). NOTE: If a gadget, no filter functions are called (use the `:selection-function` of the gadget), the `realval` is ignored, and the `:changed-values` slot is not valid. Useful gadgets are described in section 3.24.8.
- If the `filter` is non-NIL, it is a function called after the user types the value (see below).
- The `realval`, if supplied, is the actual value the `stringval` represents (e.g. if the real values are not strings). If `stringval` is a list of strings, then `realval` should be a list of the same length.
- If supplied, the `comment` is displayed after the label. It can be any string, and will be displayed after the slot label. Typical uses would be to give legal values (e.g.: "(1..20)").

`:default-filter` - If there is no filter on an individual item, then the global default-filter function is called when the user finishes editing. See below. The default filter does nothing.

`:v-spacing` - Vertical space between the items. Default = 1

`:pixel-margin` - Multiple-valued items are represented as an aggrelist, so this determines the maximum pixel value of an item, before wrapping to the next line. Note that this does *not* affect single valued items. Default: NIL

`:rank-margin` - Same as `:pixel-margin`, but is a count of the number of values. Default: NIL

`:multi-line-p` - Whether the user can enter multi-line strings, which means that `return` does not exit a field, but makes a new line. Default: NIL.

`:select-label-p` - Whether pressing on the label (with any mouse button) causes the item to be selected. Default: NIL.

`:label-select-event` - If you want to make the labels selectable, you can specify which mouse event to use in the slot `:label-select-event`.

`:label-selected-func` - Called with *(gadget label-obj label)* when a label is selected.

`:select-value-p` - Whether pressing on the value (with the right button) causes the value to be selected. NOTE: Values which are specified as gadgets cannot be selected. Default: NIL.

`:value-selected-func` - Called when a value is selected with *(gadget value-obj value label)* where label is the label of that field.

`:single-select-p` - Whether a single label or value can be selected (T) or multiple fields can be selected (NIL). This is only relevant if one or both of `:select-label-p` or `:select-value-p` is non-NIL. Default: NIL.

Read-only (output) slots:

`:label-selected` - Will be set with a list of the selected label objects. Call

`Get-Val-For-PropSheet-Value` to get label name from the label object.

`:value-selected` - Will be set with a list of the selected value objects. Call `Get-Val-For-PropSheet-value` on an obj to get the value and label from the value object.

`:value` - List of all the slots and their (filtered) values. For example:

```
( (label1 value1) (label2 value2) ...)
```

.

`:changed-values` - List of the slots that have changed, as:

```
( (label1 value1) (label2 value2) )
```

This slot is not kept valid if a gadget is used as an item.

Filter functions:

The filter functions allow the program to convert the string values to the appropriate form. The displayed string and the "real" value are stored separately, so they can be different. Filter functions are defined as:

```
(lambda (prop-sheet-gadget label value-obj new-str old-str))
```

The *index* is used for multi-valued slots, and otherwise is zero. The *value-obj* is the actual object used to display the string, and will be needed only by hackers. The filter function can return the value to use (modified *new-str*, not necessarily a string) or it can return three values:

```
(new-val in-valid-p new-str)
```

where *new-val* is a value (not necessarily a string) to use, *in-valid-p* is T if the new-str value is invalid (bad), in which case the *new-str* is still used, but it is shown in italic. If *new-str* is returned, then it is displayed instead of what the user typed (for example if the filter function expands or corrects the typed value).

An example of a custom filter function is shown in section 3.24.9.

3.24.3. Prop-Sheet-For-Obj

When you want to display a property sheet for a Garnet object, you can use `prop-sheet-for-obj`. The prop-sheet can directly access the `:parameters` list of a Garnet object, which is a list of the slots normally customizable for the object. You can also display and modify slots of *multiple* objects simultaneously. Gilt makes heavy use of many features in this prop-sheet.

```
(create-instance 'gg:Prop-Sheet-For-Obj gg:prop-sheet
    (:maybe-constant '(:left :top :obj :slots :eval-p :set-immediately-p
                       :v-spacing :multi-line-p :select-label-p
                       :label-selected-func :label-select-event :visible
                       :select-value-p :value-selected-func :single-select-p
                       :type-gadgets :union? :error-gadget))
    (:left 5)
    (:top 5)
    (:obj NIL)    ; a single obj or a list of objects
    (:slots NIL) ; list of slots to show. If NIL, get from :parameters
    (:union? T)   ;if slots is NIL and multiple objects, use union or intersection of :parameters?

    (:eval-p T)   ; if T, then evaluates what the user types.  Use T for
                  ; graphical objects.  If NIL, then all the values will be strings.
    (:set-immediately-p T) ; if T then sets slots when user hits return, else doesn't
                           ; ever set the slot.
    (:type-gadgets NIL) ; descriptor of special handling for types
    (:error-gadget NIL) ; an error gadget to use to report errors.

    ;;  plus the rest of the slots also provided by prop-sheet

    (:v-spacing 1)
    (:pixel-margin NIL)
    (:rank-margin NIL)
    (:multi-line-p NIL)      ; T if multi-line strings are allowed
    (:select-label-p NIL)    ; T if want to be able to select the labels
    (:label-select-event :any-mousedown)
    (:label-selected-func NIL)
    (:select-value-p NIL)    ; if want to be able to select the values
    (:value-selected-func NIL)
    (:single-select-p NIL) ; to select more than one value or label

   ; Read-only slots
    (:label-selected NIL)    ; set with the selected label objects (or a list)
    (:value-selected NIL)    ; set with the selected value objects (or a list)
    (:value ...)   ; list of pairs of all the slots and their (filtered) values
    (:changed-values NIL)) ; only the values that have changed
```

```
LEFT:  150
TOP:  10
WIDTH:  50
HEIGHT:  40
LINE-STYLE:  OPAL:LINE-2
FILLING-STYLE:  OPAL:GRAY-FILL
```

Figure 3-10: Example of a property sheet for an object (the object is shown at the upper left).

The loader for `prop-sheet-for-obj` is "prop-sheet-loader".

Customizable slots:

`:left`, `:top` - Position of the gadget. Default: 0,0

`:obj` - The KR object or list of objects to be displayed. If this slot contains a list of objects, then if multiple objects share a slot which is displayed, then the value from the first object is shown. If the values from multiple objects differ, then the slot value is shown in italics. If the user edits the value, then it is set into each object which has that slot in its `:parameters` list.

`:error-gadget` - An error-gadget may be placed in this slot. Type checking is performed before setting a slot, and any errors are reported in this error gadget. If there is no error gadget, then the error message is simply not displayed, but a beep is sounded and the slot value is shown in italics.

`:slots` - The list of slots of the object to view. Default value is NIL, which means the prop-sheet should get the list of slots from the `:parameters` slot of the object being edited (see `:union?`). When relying on `:parameters`, the property sheet will use the `Horiz-Choice-List` gadget for slots of type `KR-boolean` and `(Member ...)` where the number of options is 5 or less (see also `:type-gadgets`). If the type of a slot has a documentation string, gotten using `kr:get-type-documentation`, then this is displayed as the slot comment field.

Alternatively, any element in the list can be a slot name or a sublist: (*slot* "*comment*" *display*):

- If the *comment* is non-NIL, it is displayed after the label.
- If the `display` parameter is supplied, it can either be:
  - A list of legal values for the slot, e.g. `'(:direction (:horizontal :vertical))`
  - A function of the form `(lambda (new-val))` which returns T if the value is bad. This function might pop up an error dialog box after testing but before returning. The slot keeps its illegal value, but it is shown in italics.
  - A gadget, in which case the `:value` slot of the gadget is set with the old value, and the `:value` slot is queried to get the final value. If gadgets are used, then `:set-immediately-p` for the property sheet should be NIL. A useful gadget is `Pop-Up-From-Icon`.

`:union?` - This affects which slots are shown for objects when their `:parameters` lists are being used. If there are multiple objects, then a value of T for this slot will display the slots that are in *any* of the objects. If the value of this slot is NIL, then only those slots that appear in *all* of the `:parameters` lists (the intersection of the lists) will be displayed. The default is T, to show the union of all `:parameters` lists.

`:eval-p` - If NIL, then the values set into the slots will be all strings. If T, then evaluates what the user types (using `Read-From-String`) and sets the result into the slot. Usually, you use T when displaying the graphical fields of graphical objects. Default=T. NOTE: Evaluating a slot may cause the interface to crash if the values are not valid.

`:set-immediately-p` - If T, then as soon as the user types CR, the object's slot is set. If NIL, some external action must set the object's slots (e.g., when using `prop-sheet-for-obj-with-OK`, the object's slots are not set until the OK button is hit). Default=T.

`:type-gadgets` - This slot is used to modify the default displays for slots from the `:parameters` list. `:Type-gadgets` contains a list which can contain the following entries:

- a slot name - this means never display this slot (omit the slot even though it is in the `:parameters` list).
- a list of `(typ gadget)` - this means whenever a slot of type `typ` is displayed in the prop-sheet, use the specified gadget. For example, Gilt uses this mechanism to display a `Pop-Up-From-Icon` for all slots which contain a font:

```
(list (g-type opal:text :font)
      (create-instance NIL gg:Pop-Up-From-Icon
        (:constant :icon-image :pop-up-function)
        (:creator-function 'Show-Font-Dialog)
        (:pop-up-function 'Pop-Up-Prop-Dialog)))
```

- a list of `(typ othertyp)` - this means whenever a slot of type `typ` is found, pretend instead that it has type `othertyp`. This is useful, for example, to map types that are complicated to ones that will generate a `member` gadget.

The slots `:v-spacing`, `:pixel-margin`, `:rank-margin`, `:multi-line-p`, `:select-label-p`, `:label-select-event`, `:label-selected-func`, `:select-value-p`, `:value-selected-func`, and `:single-select-p` are the same as for the `prop-sheet` gadget.

Read-only (output) slots (same as `Prop-Sheet`)

`:label-selected`

`:value-selected`

`:value`

`:changed-values`

3.24.4. Useful Functions

`gg:ReUsePropSheet` *prop-sheet-gadget new-items* [*Function*]

`ReUsePropSheet` allows you to re-use an old `prop-sheet` or a `prop-sheet-with-OK` gadget with a new set of values, which is much more efficient than destroying and creating a new `prop-sheet`. NOTE: it is NOT sufficient to simply `s-value` the `:items` slot. If you plan to reuse property sheets, do not declare the `:items` slot constant.

`gg:ReUsePropSheetObj` *prop-sheet-for-obj* `&optional` *obj slots* [*Function*]

`ReUsePropSheetObj` allows a `prop-sheet-for-obj` or `prop-sheet-for-obj-with-OK` gadget to be re-used. If the new *obj* and *slots* are *not* supplied, then they should be set into the object before this function is called. NOTE: it is NOT sufficient to simply `s-value` the `:obj` and `:slots` slot.

`gg:Get-Val-For-Propsheet-Value (label-or-value-obj)` [*Function*]

The `Get-Val-For-PropSheet-Value` function returns the label when a label is passed in, or for a value-obj, returns multiple values: `value label`, where *label* is the label (name, not object) of that field.

If you want to change the value of a property sheet item without regenerating a new property sheet, you can use the new function `Set-Val-For-PropSheet-Value`. This takes the form:

`gg:Set-Val-For-PropSheet-Value` *label-or-value-obj new-value* [*Function*]

The *label-or-value-obj* parameter is the object used by the property-sheet to represent the field.

3.24.5. Prop-Sheet-With-OK

The next set of gadgets combine property sheets with OK, Apply and Cancel buttons. There are two pairs: one for Garnet look-and-feel gadgets, and one for Motif look-and-feel gadgets (see section 4.16 for the Motif version).

```
(create-instance 'gg:Prop-Sheet-With-OK opal:aggregadget
    (:maybe-constant '(:left :top :items :default-filter :ok-function
                       :apply-function :Cancel-Function :v-spacing
                       :multi-line-p :select-label-p  :visible
                       :label-selected-func :label-select-event
                       :select-value-p :value-selected-func :single-select-p))
   ; Customizable slots
    (:left 0) (:top 0)
    (:items NIL)
    (:default-filter 'default-filter)
    (:OK-Function NIL)
    (:Apply-Function NIL)
    (:Cancel-Function NIL)
    (:v-spacing 1)
    (:pixel-margin NIL)
    (:rank-margin NIL)
    (:multi-line-p NIL)  ; T if multi-line strings are allowed
    (:select-label-p NIL)  ; T if want to be able to select the entries
    (:label-select-event :any-mousedown)
    (:label-selected-func NIL)
    (:select-value-p NIL)
    (:value-selected-func NIL)
    (:single-select-p NIL)

   ; Read-only slots
    (:label-selected ...)
    (:value-selected ...)
    (:value ...)
    (:changed-values ...))
```

The `prop-sheet-with-OK` gadget is just the `prop-sheet` gadget with Garnet text buttons for OK, Apply, and Cancel.

The loader for `prop-sheet-with-OK` is "prop-sheet-win-loader".

Customizable slots

`:OK-Function` - Function called when the OK button is hit. Defined as:

```
(lambda (Prop-Sheet-With-OK-gadget)
```

Typically, this would do something with the values gotten from

```
(gv Prop-Sheet-With-OK-gadget :values)  or
(gv Prop-Sheet-With-OK-gadget :changed-values).
```

If you use the `Pop-Up-Win-For-Prop` functions, then the window will be removed before the `OK-function` is called, so you do not have to worry about the window.

`:Apply-Function` - Function called when the Apply button is hit. Defined as:

```
(lambda (Prop-Sheet-With-OK-gadget)
```

Typically, this would do something with the values gotten from

```
(gv Prop-Sheet-With-OK-gadget :values)  or
(gv Prop-Sheet-With-OK-gadget :changed-values).
```

`:Cancel-Function` - Function called when Cancel button is hit. Defined as:

```
(lambda (Prop-Sheet-With-OK-gadget))
```

Programmers typically would not use this. If you use the `Pop-Up-Win-For-Prop` functions, then the window will be removed before the `Cancel-function` is called, so you do not have to worry about the window.

The rest of the slots are the same as for `prop-sheet`.

3.24.6. Prop-Sheet-For-Obj-With-OK

```
(create-instance 'gg:Prop-Sheet-For-Obj-With-OK prop-sheet-with-OK
    (:maybe-constant '(:left :top :obj :slots :eval-p :ok-function
                       :apply-function :Cancel-Function :v-spacing
                       :multi-line-p :select-label-p :visible
                       :label-selected-func :label-select-event
                       :select-value-p :value-selected-func :single-select-p))
   ; Customizable slots
    (:OK-Function NIL)
    (:Apply-Function NIL)
    (:Cancel-Function NIL)
    (:left 0) (:top 0)
    (:obj NIL)     ; a single obj or a list of objects
    (:slots NIL)  ; list of slots to show. If NIL, get from :parameters
    (:eval-p T)    ; if T, then evaluates what the user types.  Use T for
                   ; graphical objects.  If NIL, then all the values will be strings.
    (:set-immediately-p T)  ; if T then sets slots when user hits return, else doesn't
                            ; ever set the slot.
    (:type-gadgets NIL) ; descriptor of special handling for types
    (:error-gadget NIL) ; an error gadget to use to report errors.

    ;;  plus the rest of the slots also provided by prop-sheet

    (:v-spacing 1)
    (:pixel-margin NIL)
    (:rank-margin NIL)
    (:multi-line-p NIL)     ; T if multi-line strings are allowed
    (:select-label-p NIL)  ; T if want to be able to select the labels
    (:label-select-event :any-mousedown)
    (:label-selected-func NIL)
    (:select-value-p NIL)  ; if want to be able to select the values
    (:value-selected-func NIL)
    (:single-select-p NIL)  ; to select more than one value or label

   ; Read-only slots
    (:label-selected NIL)  ; set with the selected label objects (or a list)
    (:value-selected NIL)  ; set with the selected value objects (or a list)
    (:value ...)    ; list of pairs of all the slots and their (filtered) values
    (:changed-values NIL))  ; only the values that have changed
```

The `prop-sheet-for-obj-with-OK` gadget is just the `prop-sheet-for-obj` gadget with Garnet text buttons for OK, Apply, and Cancel.

The loader for `prop-sheet-for-obj-with-OK` is "prop-sheet-win-loader".

Given a list of slots for a KR object, displays the values and allows them to be edited. The labels and values can optionally be selectable. Sets the object's slot only when OK or Apply is hit. (So `:set-immediately-p` is always NIL).

Customizable slots

`:OK-Function` - Function called when the OK button is hit. Defined as:

```
(lambda (Prop-Sheet-For-Obj-With-OK-gadget))
```

Since this gadget will set the slots of the object automatically when OK is hit (before this function is called) and the window visibility is handled automatically, programmers rarely need to supply a function here.

`:Apply-Function` - Function called when the Apply button is hit. Defined as:

```
(lambda (Prop-Sheet-For-Obj-With-OK-gadget))
```

Since this gadget will set the slots of the object automatically when Apply is hit (before this function is called), programmers rarely need to supply a function here.

`:Cancel-Function` - Function called when Cancel button is hit. Defined as:

```
(lambda (Prop-Sheet-For-Obj-With-OK-gadget)
```

Since the window visibility is handled automatically, programmers rarely need to supply a function here.

3.24.7. Useful Functions

`gg:Pop-Up-Win-For-Prop` *prop-gadget-with-ok left top title* `&optional` *modal-p* [*Function*]

Given an existing gadget of any of the "OK" types, this function pops up a window which will show the property sheet, and will go away when the user hits either "OK" or "Cancel". The window is allocated by this function to be the correct size. When the *modal-p* parameter is T, then interaction in all other Garnet windows will be suspended until the user clicks either the "OK" or "Cancel" button in this window. This function can be called many times on the same gadget, which is much more efficient than allocating a new gadget and window each time. To change the items or object before redisplaying, use one of the functions below.

`gg:Pop-Up-Win-Change-Items` *prop-gadget-with-ok new-items left top title* `&optional` *modal-p* [*Function*]

Given an existing gadget, `Pop-Up-Win-Change-Items` sets the items field of the gadget to the specified value, and then pops up a window displaying that property sheet. (This function calls `ReUsePropSheetObj` automatically). (Note: if you want to pop up a `Prop-Sheet-With-OK` or `Motif-Prop-Sheet-With-OK` gadget without changing the *items* field, you can simply pass it to `Pop-Up-Win-For-Prop`.

`gg:Pop-Up-Win-Change-Obj` *prop-obj-gadget-with-ok obj slots left top title* `&optional` *modal-p* [*Function*]

Given an existing gadget, `Pop-Up-Win-Change-Obj` sets the *obj* and *slot* fields of the gadget to the specified values, and then pops up a window displaying that property sheet. (This function calls `ReUsePropSheetObj` automatically). (Note: if you want to pop up a `Prop-Sheet-For-Obj-With-OK` or `Motif-Prop-Sheet-For-Obj-With-OK` gadget without changing the *obj* and *slot* fields, you can simply pass it to `Pop-Up-Win-For-Prop`.

3.24.8. Useful Gadgets

This section describes two gadgets that are useful in property sheet fields as the values. Both of these gadgets are shown in Figure 4-6.

3.24.8.1. Horiz-Choice-List

The `horiz-choice-list` displays the choices and allows the user to pick one with the left mouse button. The choices can be strings or atoms.

```
(create-instance 'gg:Horiz-Choice-List opal:aggregadget
   (:maybe-constant '(:left :top :items))
   ; Customizable slots
   (:left 0)  ; left and top are set automatically when used in a prop-sheet
   (:top 0)
   (:items '("one" "two" "three"))  ; the items to choose from
   ; Input and output slot
   (:value NIL)  ; what the user selected
   )
```

The loader for `Horiz-Choice-List` is "prop-values-loader", although it is automatically loaded when you load a property sheet.

The `Horiz-Choice-List` is automatically used when you list a set of legal values for the display parameter for a `prop-sheet-for-obj`.

3.24.8.2. Pop-Up-From-Icon

The `Pop-Up-From-Icon` displays a small icon, and if the user hits on it, then a function is called which can pop-up a dialog box or menu to make the choice.

```
(create-instance 'gg:Pop-Up-From-Icon opal:aggregadget
  (:maybe-constant '(:left :top :icon-image :pop-up-function))
  ; Customizable slots
  (:left 0)  ; left and top are set automatically when used in a prop-sheet
  (:top 0)
  (:icon-image pop-up-icon)  ; you can replace with your own picture
  (:pop-up-function NIL))    ;put a function here to pop-up the menu or whatever
```

The loader for `Pop-Up-From-Icon` is "prop-values-loader", although it is automatically loaded when you load a property sheet.

The `pop-up-function` is called when the user presses with the left button and then releases over the icon. It is called as follows:

```
(lambda(pop-up-from-icon-gadget))
```

It should stuff its results into the `:value` field of that gadget. See the manual on Gilt for some functions that are useful for popping up dialog boxes and menus.

3.24.9. Property Sheet Examples

First, an example filter function, which checks if value is a number, and if it is between 1 and 20.

```
(defun string-to-num-filter (prop-gadget label index value-obj new-str old-str)
  (declare (ignore prop-gadget label index value-obj))
  (let* ((sym (read-from-string new-str))
         (number (when (integerp sym) sym)))
    (if (and number (>= number 1) (<= number 20))
        ; then OK, return the converted number
        (values number NIL new-str)
        ; else bad, return original string and T to say invalid
        (progn
          (inter:beep) ; first, beep
          (values new-str T new-str)))))
```

Now, we will use that filter function in a property sheet. This code creates the property sheet shown in Figure 3-9 in section 3.24.2. It contains three regular lines, a slot using a gadget, and then a slot with a filter function and a comment.

```
(create-instance 'PROP1 garnet-gadgets:prop-sheet
   (:items `((:color "Red")
             (:height "34")
             (:status "Nervous")
             (:direction ,(create-instance NIL garnet-gadgets:horiz-choice-list
                             (:items '("up" "down" "diagonal"))))
             (:range "1" ,#'string-to-num-filter 1 "(1..20)"))))
```

Finally, a Motif look and feel property sheet for an object with OK, Apply and Cancel buttons in it. The `my-rectangle1` object is only changed when OK or Apply is hit. The resulting window is shown in Figure 4-6.

```
(create-instance 'MY-OBJ-PROP gg:motif-prop-sheet-for-obj-with-OK
  (:left 0)
  (:top 0)
  (:obj MY-RECTANGLE1)
  (:slots `(:left ; first four slots are normal
            :top
            :width
            :height
            (:quality (:good :medium :bad)) ;list of options
            ; next two slots use pop-up icon gadgets
            (:line-style ,(create-instance NIL gg:pop-up-from-icon
                             (:pop-up-function #'Line-Style-Pop-Up)))
            (:filling-style ,(create-instance NIL gg:pop-up-from-icon
                                (:pop-up-function #'Fill-style-pop-up))))))
```

3.25. Mouseline

There are two new gadgets that will show a help string attached to any object. The string can be shown in a fixed location in a window using the `MouseLine` gadget, and therefore is like the **mouse documentation line** on Symbolics Lisp machines (sometime called the ‘‘mode line’’ or ‘‘who line’’). Alternatively, the help string can pop up in a window using the `MouseLinePopup` gadget, and therefore be like **Balloon Help** in the Macintosh System 7. You can also control whether the string appears immediately or only after the mouse is over an object for a particular period of time.

An example of the use of the two mouseline gadgets is `gg:mouseline-go` which is at the end of the mouseline.lisp file. The standard demos-controller which you get when you load garnet-demos-loader also uses the MouseLinePopup gadget to show what the different demos do.

Note: the mouseline gadget is implemented in a rather inefficient manner. It has the potential to significantly slow down applications, especially when the delay feature is used (:wait-amount non-zero). If this proves to be a big problem in practice, please let us know.

Note 2: the delay feature is implemented with multiple processes, which are only supported in Allegro and Lucid lisp.

3.25.1. MouseLine gadget

```
(create-instance 'gg:MouseLine opal:aggregadget
  (:left 5)
  (:top (o-formula (- (gvl :window :height)  ; default is bottom of window
                      (gvl :label :height)
                      5)))
  (:windows (o-formula (gvl :window)))  ; default is the window containing the mouseline gadget
  (:wait-amount 0)   ; how long to wait before displaying the string
```

The loader file for the MouseLine is mouseline-loader.

You create an instance of the mouseline gadget and add it to a window. By default it is positioned at the bottom left, but you can override the :top and :left to position it where-ever you want. Once created, the string will display the value of the :help-string field for any object the mouse is over in the window or windows specified in the :windows slot. By default :windows is only the window that the mouseline gadget is in, but it can be any list of windows, or T for all interactor windows.

The gadget first looks at the leaf object under the mouse, and if that does not have a help-string, then its parent (aggregate) is looked at, and so on. The lowest-level help string found is displayed in the string. The string can contain newlines but not font information (the display is a opal:multi-text not a opal:multifont-text). Of course, the :help-string slot can contain a formula, which might, for example, generate a different string when a gadget is disabled explaining why.

If the mouseline gadgets catch on, we might provide a way to specify the help-strings as part of the standard :items protocol for gadgets, but for now you need to s-value the :help-string slots directly. See the demos-controller for how this might be done.

If the :wait-amount slot is non-zero, then it is the number of seconds the mouse must remain over an object before the mouseline string is displayed. This feature relies on the animation-interactor which uses the multi-process mechanism in Lisp, **so the :wait-amount is only currently available in Lucid, Allegro, and LispWorks.**

3.25.2. MouseLinePopup gadget

```
(create-instance 'gg:MouseLinePopup opal:aggregadget
  (:start-event :SHIFT-CONTROL-META-LEFTDOWN)
  (:windows (o-formula (gvl :window))) ; default is the window containing the mouseline gadget
  (:wait-amount 3) ; how long to wait before displaying string
```

The loader file for the MouseLinePopup is mouseline-loader.

This displays the same help-string as the mouseline gadget above, but the string is displayed in a window which pops up at the mouse. Therefore it is like ‘‘Balloon Help’’ in the Macintosh System 7. The window is just big enough for the string, and it goes away when you move off of the object. The :wait-amount determines how long in seconds you must keep the mouse over the object before the window appears.

3.26. Standard Edit

There are a number of editing functions that are shared by most graphical editors. The file `standard-edit.lisp` supplies many of these functions in a manner that can probably be used by your graphical editors without change. They support such operations such as cut, copy, paste, delete, duplicate, group, ungroup, refresh, to-top, to-bottom, etc. These functions are designed to work with the `Multi-Graphics-Selection` gadget, and can be invoked from buttons, menus, or a menubar. The `standard-edit` functions are currently used by GarnetDraw, Gilt and Marquise. You don't have to use all the functions in an application. For example, Gilt does not support grouping and ungrouping. (If you find that changing a `standard-edit` routine will allow it to be useful to your application, let us know.)

The `standard-edit` routines can be loaded using `(garnet-load "gg:standard-edit-loader")`.

3.26.1. General Operation

The `standard-edit` routines assume that the graphical objects that are to be edited are all in a single aggregate in a single window (extensions to handle multiple windows are planned, but not in place yet). The routines are tightly tied to the design of the `Multi-Graphics-Selection` gadget. For example, most routines determine which objects to operate on by looking at the current selection, and many change the selection.

`Standard-edit` determines how to edit objects by looking at various slots. The slots listed below are set in the selected objects, not in the selection gadget itself. Most Garnet prototypes already contain the correct default values:

- `:line-p` - if non-NIL, then the object is controlled by a `:points` list of 4 values. True by default for `opal:line` and `gg:arrow-lines`.
- `:polygon-p` - if non-NIL, then the object is controlled by a `:point-list` list of multiple values. True by default for `opal:polylines`.
- `:group-p` - if non-NIL, then the object is a group of objects that the user might be able to get the parts of. True by default for `opal:aggregadgets`. If you allow high-level objects to be added in your editor (e.g., gadgets like buttons), and you supply the `Standard-Ungroup` command, you should set the `:group-p` slot of any objects you don't want the user to ungroup to be NIL.
- `:grow-p` - whether the object can change size or not.

If the object has `:line-p` and `:polygon-p` both NIL, then it is assumed to be controlled by a `:box` slot.

The various routines find information they need by looking in a special slots of the gadget that invokes them. This means that all routines must be invoked from the same gadget set, for example, the same `menubar` or `motif-button-panel`.

3.26.2. The Standard-Edit Objects

The `gg:Clipboard-Object` holds the last object that was cut or copied. It also contains some parameters used for pasting and duplicating the objects. Each application can have its own clipboard, or a set of applications can share a clipboard to allow cut and paste among applications. For example, GarnetDraw and Gilt both share the same clipboard, so you can cut and paste objects between the two applications. By default, all applications share the one `gg:Default-Global-Clipboard`.

Note that this does *not* use the X cut buffer, since there is no standard way to copy graphics under X.

```
(create-instance 'gg:Clipboard-Object NIL
  (:value NIL)
  (:x-inc-amt NIL)  ; Offset for duplicate.  If NIL, then uses 10
  (:y-inc-amt NIL))

(create-instance 'gg:Default-Global-Clipboard gg:Clipboard-Object)
```

The `Default-Global-Clipboard` is used by default, and allows objects to be copied from one Garnet application to another.

3.26.3. Standard Editing Routines

`gg:Standard-Initialize-Gadget` *gadget selection-gadget agg-of-items* `&key` *clipboard undo-delete?* [*Function*]

This routine must be called once before any of the others are invoked. Typically, you would call this after the editor's windows and objects are created. It takes the `gadget` that is going to invoke the standard-edit routines (e.g., a menubar), the selection gadget that is used to select objects in the graphics editor, and the aggregate that holds the items created in the graphics editor. If you do not supply a `clipboard` object, then `Default-Global-Clipboard` will be used.

Unfortunately, there is not yet a global undo facility, but you can support undoing just the delete operations. The `undo-delete?` flag tells standard-edit whether you want this or not. If non-NIL, then deleted objects are never destroyed, they are just saved in a list.

`gg:Standard-NIY` *gadget* `&rest` *args* [*Function*]

Useful for all those functions that are Not Implemented Yet. It prints "Sorry, Not Implemented Yet" in the Lisp listener window and beeps.

`gg:Standard-Delete` *gadget* `&rest` *args* [*Function*]

Deletes all the selected objects. Makes there be no objects selected.

`gg:Standard-Delete-All` *gadget* `&rest` *args* [*Function*]

Deletes all the objects. Makes there be no objects selected.

`gg:Standard-Undo-Last-Delete` *gadget* `&rest` *args* [*Function*]

If you have initialized standard-edit with `Undo-delete?` as non-NIL, then this function will undo the last delete operation. The objects brought back are selected.

`gg:Standard-To-Top` *gadget* `&rest` *args* [*Function*]

Moves the selected objects to the top (so not covered). They stay selected.

`gg:Standard-To-Bottom` *gadget* `&rest` *args* [*Function*]

Moves the selected objects to the bottom (so covered by all other objects). They stay selected.

`gg:Standard-Refresh` *gadget* `&rest` *args* [*Function*]

Simply redraws the window containing the objects using `(opal:update win T)`.

`gg:Standard-Select-All` *gadget* `&rest` *args* [*Function*]

Causes all of the objects to be selected.

`gg:Standard-Cut` *gadget* `&rest` *args* [*Function*]

Copies the selected objects into the clipboard's cut buffer, and then removes them from the window. Afterwards, there will be no selection.

`gg:Standard-Copy` *gadget* `&rest` *args* [*Function*]

Copies the selected objects into the clipboard's cut buffer, but leaves them in the window. The selection remains the same.

`gg:Standard-Paste-Same-Place` *gadget* `&rest` *args* [*Function*]

Pastes the objects in the clipboard into the window at the same place from which they were cut. Pasting the same objects multiple times will give multiple copies, all in the same place. An application will typically provide either `Standard-Paste-Same-Place` or `Standard-Paste-Inc-Place` as the ‘‘paste’’ operation. The new objects will be selected.

`gg:Standard-Paste-Inc-Place` *gadget* `&rest` *args* [*Function*]

Pastes the objects in the clipboard into the window offset from where they were cut. Pasting the same objects multiple times will give multiple copies, each offset from the previous. The offset amount is determined by the `:x-inc-amt` and `:y-inc-amt` slots of the clipboard object, or, if NIL, then 10 is used. The new objects will be selected.

`gg:Standard-Duplicate` *gadget* `&rest` *args* [*Function*]

Makes a copy of the selected objects, and places them back into the window, offset from the previous objects by `:x-inc-amt` and `:y-inc-amt` (or 10 if these are NIL). The new objects will be selected.

`gg:Standard-Group` *gadget* `&rest` *args* [*Function*]

Creates an `aggregadget` and puts the selected objects into it. The `Multi-Graphics-Selection` gadget will then operate on the group as a whole, and will not let parts of it be manipulated (like MacDraw, but unlike Lapidary). The group (aggregadget) object will be selected.

`gg:Standard-UnGroup` *gadget* `&rest` *args* [*Function*]

Goes through all the selected objects, and for any that have the `:group-p` slot non-NIL, removes all the components from that aggregate and adds the objects directly to the parent of the group. `:Group-p` is true by default for `opal:aggregadgets`. If you allow high-level objects to be added in your editor (e.g., gadgets like buttons), and you supply the `Standard-Ungroup` command, you should set the `:group-p` slot to be NIL for any objects you don't want the user to ungroup.

3.26.4. Utility Procedures

`gg:Sort-Objs-Display-Order` *objs draw-agg* [*Function*]

For many operations, it is important to operate on the objects in display order, rather than in the order in which the objects were selected. `Sort-Objs-Display-Order` takes a list of objects (*objs*) and an aggregate that contains them (*draw-agg*) and sorts the objects so they are in the same order as in *draw-agg*. The procedure returns a <u>copy</u> of the list passed in, so it is safe to supply the `:value` of the `Multi-Graphics-Selection` gadget, for example.

`gg:Is-A-Motif-Background` *obj* [*Function*]

Tests whether the specified object is a `Motif-Background` object. This procedure is safe even if the Motif gadgets have not been loaded.

`gg:Is-A-Motif-Rect` *obj* [*Function*]

Tests whether the specified object is a `Motif-Rect` object. This procedure is safe even if the Motif gadgets have not been loaded.

4. The Motif Gadget Objects

The Motif gadgets in the Gadget Set were designed to simulate the appearance and behavior of the OSF/Motif widgets. They are analogous to the standard gadgets of Chapter 3, and many of the customizable slots are the same for both sets of gadgets.

As in the previous chapter, the descriptions of the Motif gadgets begin with a list of customizable slots and their default values (any of which may be ignored). The `motif-gadget-prototype` object which occurs in the definition of each Motif gadget is just an instance of an `opal:aggregadget` with several color, filling-style, and line-style slot definitions used by all Motif gadgets.

The Motif gadgets have been implemented to appear on either color or black-and-white screens without changes to the instances. The `:foreground-color` slot is used to compute filling-styles internally on a color screen, and it is ignored on a black-and-white screen. Figure 4-1 shows how a few of the Motif gadgets look on each type of screen.

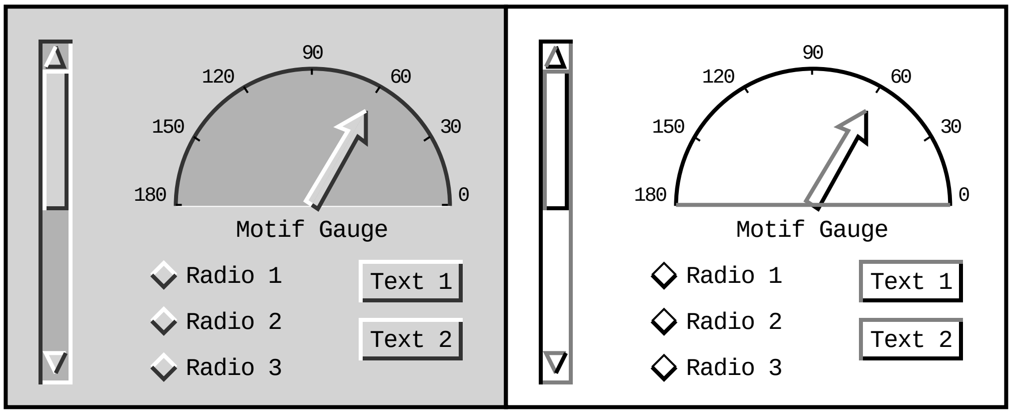


Figure 4-1: Motif style gadgets on color and black-and-white screens

4.1. Useful Motif Objects

In order to facilitate the construction of interfaces containing Motif gadgets, Garnet exports some miscellaneous objects that are commonly found in Motif. The objects described in this section are defined in the "motif-parts" file (automatically loaded with all Garnet Motif-style "-loader" files).

4.1.1. Motif Colors and Filling Styles

In each Motif gadget, there is a slot for the color of the gadget. The `:foreground-color` is the color that should be shown in the foreground of the gadget (i.e., the part of the gadget that does not appear recessed). The background, shadow, and highlight colors for the gadget are computed internally based on the `:foreground-color` given.

The default `:foreground-color` for the gadgets is `opal:motif-gray`, but the user may provide any instance of `opal:color` in the slot. Additionally, Opal provides the following colors for use with the Motif gadgets. The associated filling styles may be of use in other objects designed by the programmer.

```
opal:motif-gray                     opal:motif-gray-fill
opal:motif-blue                     opal:motif-blue-fill
opal:motif-green                    opal:motif-green-fill
opal:motif-orange                   opal:motif-orange-fill
opal:motif-light-gray               opal:motif-light-gray-fill
opal:motif-light-blue               opal:motif-light-blue-fill
opal:motif-light-green              opal:motif-light-green-fill
opal:motif-light-orange             opal:motif-light-orange-fill
```

When the Motif gadgets are used on a black-and-white monitor, the gadgets ignore the `:foreground-color` slot and internally compute reasonable filling-styles that are black, white, or Opal halftones.

4.1.2. Motif-Background

```
(create-instance 'gg:Motif-Background opal:rectangle
   (:foreground-color opal:motif-gray))
```

In order to simulate the Motif three-dimensional effect in an interface, there should be a gray background in a window containing Motif-style gadgets. Garnet provides two ways to achieve this effect. You could add an instance of the `motif-background` object to the window, which is a rectangle whose dimensions conform to the size of the window in which it appears.

Alternately, you could supply the `:background-color` of your window with an appropriate Opal `color` object (like `opal:motif-gray`). This is generally more efficient, since it is faster to redraw a window with its background color than to redraw a rectangle that occupies the entire window.

NOTE: If you choose to use the `motif-background` object, it is essential that the instance be added to the top-level aggregate before any other Garnet object. This will ensure that the background is drawn behind all other objects in the window.

Of course, the `:foreground-color` of the `motif-background` instance or the `:background-color` of the window should be the same as the colors of all the Motif gadgets in the window.

4.1.3. Motif-Tab-Inter

```
(create-instance 'gg:Motif-Tab-Inter inter:button-interactor
   (:window NIL)
   (:objects NIL)
   (:rank 0)
   (:continuous NIL)
   (:start-where T)
   (:start-event '(#\tab :control-tab))
   (:waiting-priority gg:motif-tab-priority-level)
   (:running-priority gg:motif-tab-priority-level)
   (:stop-action #'(lambda (interactor obj-over) ...))
   (:final-function NIL))
```

Each Motif gadget has the ability to be operated by the keyboard as well as the mouse. In traditional Motif interfaces, the keyboard selection box is moved within each gadget with the arrow keys, and it is moved among gadgets with the tab key (i.e., one gadget's keyboard selection is activated while the previous gadget's keyboard selection is deactivated). The keyboard interface can be manually activated by setting a Motif gadget's `:keyboard-selection-p` to T, but the bookkeeping becomes formidable when there are a large number of Motif gadgets on the screen and their keyboard status is changing. Thus, Garnet provides the `motif-tab-inter` which handles the bookkeeping among multiple Motif gadgets.

To use the motif-tab-inter, create an instance with a list of the Motif gadgets on which to operate in the :object slot and the window of the objects in the :window slot. Usually, these are the only two slots that will need to be set.

Repeatedly hitting the tab key (or simultaneously hitting control and tab will cause the keyboard selection to cycle through the list of objects. Specifically, hitting the tab key causes the :rank of the motif-tab-inter to be incremented, and the interactor checks the :active-p slot of the next object in the :object list. If the result is T, then that object's :keyboard-selection-p slot is set to T. Otherwise, the :rank is incremented again and the next object is checked.

The :active-p slots of the "continuous" Motif gadgets -- the scroll bars, slider, and gauge -- all default to T, while the :active-p slots of the Motif buttons and menu depend on the items in the :inactive-items list.

The :running-priority and :waiting-priority of the motif-tab-inter are both set to be motif-tab-priority-level, which is a higher priority than the default interactor priority levels (but lower than the error-gadget's error-priority-level). This allows the motif-tab-inter to be used at the same time as the inter:text-interactor (as in the motif-scrolling-labeled-box).

The function in the :final-function slot is executed whenever the current selection changes. It takes the parameters

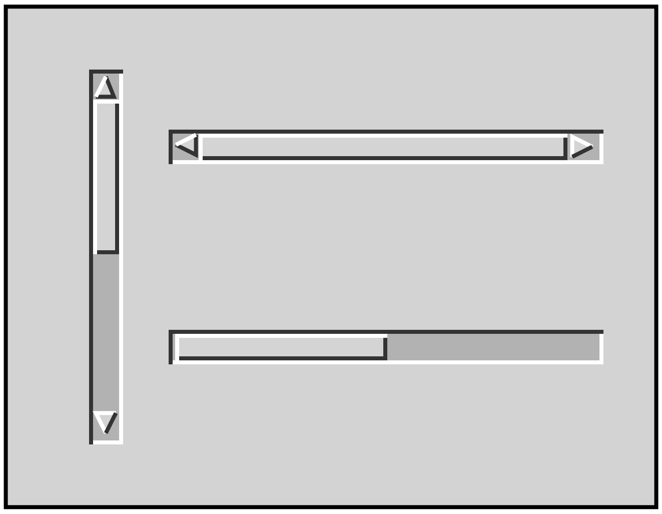
```
(lambda (inter new-object))
```

Examples of the motif-tab-inter in use may be found in demo-motif and in all three Motif button demos.

4.2. Motif Scroll Bars

```
(create-instance 'gg:Motif-V-Scroll-Bar gg:motif-gadget-prototype
   (:maybe-constant '(:left :top :width :height :val-1 :val-2 :scr-incr
                      :page-incr :scr-trill-p :percent-visible :scroll-p
                      :foreground-color :visible))
   (:left 0)
   (:top 0)
   (:width 20)
   (:height 200)
   (:val-1 0)
   (:val-2 100)
   (:scr-incr 1)
   (:page-incr 5)
   (:scr-trill-p T)
   (:percent-visible .5)
   (:scroll-p T)
   (:keyboard-selection-p NIL)
   (:foreground-color opal:motif-gray)
   (:value (o-formula ...))
   (:active-p T)
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

```
(create-instance 'gg:Motif-H-Scroll-Bar gg:motif-gadget-prototype
   (:maybe-constant '(:left :top :width :height :val-1 :val-2 :scr-incr
                      :page-incr :scr-trill-p :percent-visible :scroll-p
                      :foreground-color :visible))
   (:left 0)
   (:top 0)
   (:width 200)
   (:height 20)
   (:val-1 0)
   (:val-2 100)
   (:scr-incr 1)
   (:page-incr 5)
   (:scr-trill-p T)
   (:percent-visible .5)
   (:scroll-p T)
   (:keyboard-selection-p NIL)
   (:foreground-color opal:motif-gray)
   (:value (o-formula ...))
   (:active-p T)
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```



The loader file for the `motif-v-scroll-bar` is "motif-v-scroll-loader". The loader file for the `motif-h-scroll-bar` is "motif-h-scroll-loader".

The Motif scroll bars allow the specification of the minimum and maximum values of a range, while the `:value` slot is a report of the currently chosen value in the range. The interval is determined by the values in `:val-1` and `:val-2`, and either slot may be the minimum or maximum of the range. The value in `:val-1` will correspond to the top of the vertical scroll bar and to the left of the horizontal scroll bar. The `:value` slot may be accessed directly by some function in the larger interface, and other formulas in the interface may depend on it. If the `:value` slot is set directly, then the appearance of the scroll bar will be updated accordingly.

The trill boxes at each end of the scroll bar allow the user to increment and decrement `:value` by the amount specified in `:scr-incr`. The designer may choose to leave the trill boxes out by setting `:scr-trill-p` to NIL.

The indicator may also be moved directly by mouse movements. Dragging the indicator while the left mouse button is pressed will change the `:value` accordingly. A click of the left mouse button in the background trough of the scroll bar will cause the `:value` to increase or decrease by `:page-incr`, depending on the location of the indicator relative to the mouse click.

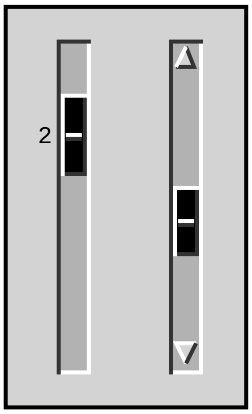
When `:keyboard-selection-p` is T, then a black-selection box is drawn around the scroll bar and the indicator can be moved with the arrow keys (uparrow and downarrow for the `motif-v-scroll-bar`, leftarrow and rightarrow for the `motif-h-scroll-bar`).

The `:percent-visible` slot contains a value between 0 and 1, and is used to specify the length of the indicator relative to the length of the trough. If `:percent-visible` is .5, then the length of the indicator will be half the distance between the two trill boxes. This feature might be useful in a scrolling menu where the length of the indicator should correspond to one "page" of items in the menu (e.g., for three pages of items, set `:percent-visible` to .33).

The slots `:scroll-p` and `:active-p` are used to enable and disable the scrolling feature of the scroll bar. When either is set to NIL, the trill boxes of the scroll bar become inactive and the indicator cannot be moved. The difference is that when `:active-p` is set to NIL, then the keyboard selection cannot be enabled.

4.3. Motif Slider

```
(create-instance 'gg:Motif-Slider gg:motif-v-scroll-bar
   (:maybe-constant '(:left :top :height :trough-width :val-1 :val-2
                      :scr-incr :page-incr :scr-trill-p :text-offset
                      :scroll-p :indicator-text-p :indicator-font
                      :foreground-color :visible))
   (:left 0)
   (:top 0)
   (:height 200)
   (:trough-width 16)
   (:val-1 0)
   (:val-2 100)
   (:scr-incr 1)
   (:page-incr 5)
   (:scr-trill-p NIL)
   (:text-offset 5)
   (:scroll-p T)
   (:indicator-text-p T)
   (:keyboard-selection-p NIL)
   (:indicator-font opal:default-font)
   (:foreground-color opal:motif-gray)
   (:value (o-formula ...))
   (:active-p T)
   (:selection-function NIL)    ; (lambda (gadget value))
   (:parts (...)))
```



The loader file for the `motif-slider` is "motif-slider-loader".

The `motif-slider` is similar to the `motif-v-scroll-bar`, except that it has a fixed-size indicator with accompanying text feedback. The mouse can be used to drag the indicator, and the arrow keys can be used when keyboard-selection is activated.

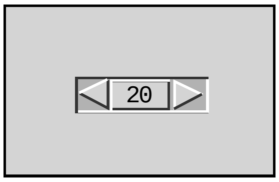
The slots `:value`, `:val-1`, `:val-2`, `:scr-incr`, `:page-incr`, `:scr-trill-p`, `:scroll-p`,

`:active-p` and `:keyboard-selection-p` all have the same functionality as in the `motif-v-scroll-bar`.

The `:trough-width` slot determines the width of the scroll-bar part of the slider. The actual `:width` of the gadget is not user-settable because of the changing value feedback width.

The current `:value` of the slider is displayed beside the trough if `:indicator-text-p` is T. The font of the indicator text is in `:indicator-font`. The distance from the indicator text to the trough is in `:text-offset`.

4.4. Motif-Trill-Device




```
(create-instance 'gg:Motif-Trill-Device gg::motif-gadget-prototype
  (:left 0) (:top 0)
  (:width 150) (:height 40)
  (:val-1 0) (:val-2 100)
  (:value 20)
  (:foreground-color opal:motif-gray)
  (:format-string "~a")
  (:value-feedback-font opal:default-font)
  (:value-feedback-p T)
  (:scroll-incr 1)
  (:selection-function NIL)    ; (lambda (gadget value))
  )
```

The loader file for the `motif-trill-device` is "motif-trill-device-loader". The demo `(gg:motif-trill-go)` is loaded by default, and shows an example of the `motif-trill-device`.

The `motif-trill-device` is a simple incrementing/decrementing gadget with trill boxes and a numerical display. The behavior is identical to the standard `trill-device` -- click on the left or right arrows to change the value, and click the left mouse button on the text to edit it.

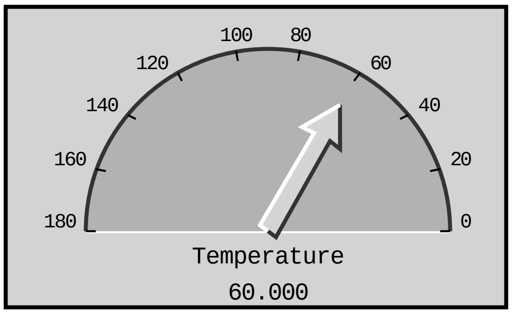
The slots `:val-1` and `:val-2` contain the upper and lower bounds for the value of the gadget. Either slot may be the minimum or maximum, and either slot may be NIL (indicating no boundary). If a value less than the minimum allowed value is entered, the value of the gadget will be set to the minimum, and analogously for the maximum. Clicking on the left trill box always moves the value closer to `:val-1`, whether that is the max or min, and clicking on the right trill box always moves the value closer to `:val-2`.

The current value of the gadget is stored in the `:value` slot, and may be set directly using `s-value`. The `:scroll-incr` slot specifies the increment for changing the value with the trill boxes. All other slots work the same as in the standard `trill-device`. See section 3.3 for more information.

The `:foreground-color` slot specifies the color of the object.

4.5. Motif Gauge

```
(create-instance 'gg:Motif-Gauge gg:motif-gadget-prototype
   (:maybe-constant '(:left :top :width :title :foreground-color :title-font
                      :value-font :enum-font :num-marks :tic-marks-p
                      :enumerate-p :value-feedback-p :text-offset :val-1 :val-2
                      :scr-incr :format-string :enum-format-string :visible))
   (:left 0)
   (:top 0)
   (:width 230)
   (:title "Motif Gauge")
   (:foreground-color opal:motif-gray)
   (:title-font opal:default-font)
   (:value-font opal:default-font)
   (:enum-font (create-instance NIL opal:font (:size :small)))
   (:num-marks 10)              ; Includes endpoints
   (:tic-marks-p T)
   (:enumerate-p T)
   (:value-feedback-p T)
   (:text-offset 5)
   (:val-1 0)
   (:val-2 180)
   (:scr-incr 5)
   (:format-string "~a")        ; How to print the feedback value
   (:enum-format-string "~a")   ; How to print the tic-mark values
   (:keyboard-selection-p NIL)
   (:value (o-formula ...))
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```



The `motif-gauge` is a semi-circular meter with tic-marks around the perimeter. As with scroll bars and sliders, this object allows the user to specify a value between minimum and maximum values. An arrow-shaped polygon points to the currently chosen value, and may be rotated either by dragging it with the mouse or by the arrow keys when keyboard selection is activated. Text below the gauge reports the current value to which the needle is pointing.

The slots `:num-marks`, `:tic-marks-p`, `:enumerate-p`, `:val-1`, `:val-2`, and `:enum-font` are implemented as in the standard Garnet sliders (see section 3.2). The value in `:val-1` corresponds to the right side of the gauge.

The title of the gauge is specified in `:title`. No title will appear if `:title` is NIL. The fonts for the title of the gauge and the current chosen value are specified in `:title-font` and `:value-font`, respectively.

If `:value-feedback-p` is T, then numerical text will appear below the gauge indicating the currently chosen value. The value in `:text-offset` determines the distance between the gauge and the title string, and between the title string and the value feedback.

The `:format-string` and `:enum-format-string` slots allow you to control the formatting of the text strings, in case the standard formatting is not appropriate. This is mainly useful for floating point numbers. The slots should each contain a string that can be passed to the lisp function `format`. The default string is `"~a"`.

Setting `:keyboard-selection-p` to T activates the keyboard interface to the `motif-gauge`. The left and right arrow keys can then be used to change the value of the gauge. The increment by which the value of the gauge changes during each press of an arrow key is in `:scr-incr`.

4.6. Motif Buttons

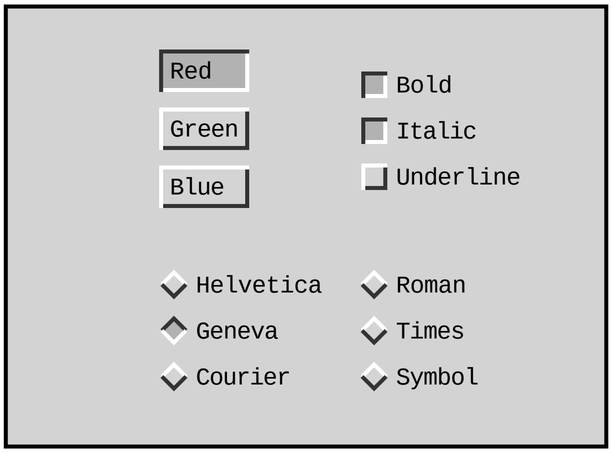


Figure 4-2: Motif Text Buttons, Check Buttons, and Radio Buttons

As with the standard Garnet buttons, the Motif buttons can be either a single, stand-alone button or a panel of buttons. Use of the Motif gadgets is identical to the use of standard Garnet buttons in the following respects (see Section 3.5).

1. All slots that can be customized in an aggrelist can be customized in the Motif button panels.
2. The `:value` slot contains the string or atom of the currently selected item (in the `motif-check-button-panel` this value is a list of selected items). In button panels, the currently selected component of the panel's aggrelist is named in the `:value-obj` slot.
3. The `:width` and `:height` of button panels are determined internally, and may not be set directly. Instead, refer to the slots `:fixed-width-size` and `:fixed-height-size`. The `:width` and `:height` slots may be accessed after the object is instantiated.
4. The `:items` slot can be either a list of strings, a list of atoms, or a list of string/function or atom/function pairs (see section 1.4.3).
5. The font in which the button labels appear may be specified in the `:font` slot.
6. Most of the buttons and button panels have a `:toggle-p` slot that controls whether buttons can become deselected. If the value of this slot is T, then clicking on a selected button deselects it. Otherwise, the button always stays selected, though the `:selection-function` and the item functions will continue to be executed each time the button is pressed.

The following slots provide additional functionality for the Motif buttons:

1. In single Motif buttons, if the `:active-p` slot is NIL, then the string of the button appears in "grayed-out" text and the button is not user selectable.
2. Analogously, the `:inactive-items` slot of the Motif button panels contains a list of strings or atoms corresponding to the members of the `:items` list. The text of each item listed in `:inactive-items` will appear "grayed-out" and those buttons will not be user selectable. If `:active-p` is set to NIL, then all items will appear "grayed-out".

3. When the slot `:keyboard-selection-p` is T, the keyboard interface to the button gadgets is activated. The arrow keys will move the selection box among the buttons in a button panel, and the space-bar will select the boxed button. The component of the button panel aggrelist currently surrounded by the selection box is named in `:keyboard-selection-obj`, and its string is in `:keyboard-selection`. Thus, the slot `:keyboard-selection` may be set with a string (or an atom, depending on the `:items` list) to put the selection box around a button. Since this slot contains a formula, the programmer may not supply an initial value at create-instance time. Instead, as with the `:value` slot, the user must first gv the `:keyboard-selection` slot and then s-value it to the desired initial value.

4. **NOTE:** When keyboard selection is activated, the space-bar is used to select buttons, while the return key is used to select items in the `motif-menu`.

4.6.1. Motif Text Buttons

```
(create-instance ’gg:Motif-Text-Button gg:motif-gadget-prototype
   (:maybe-constant ’(:left :top :text-offset :active-p :string :toggle-p :font
                      :final-feedback-p :foreground-color :visible))
   (:left 0)
   (:top 0)
   (:text-offset 5)
   (:active-p T)
   (:string "Motif Text Button")
   (:font opal:default-font)
   (:final-feedback-p NIL)
   (:toggle-p T)
   (:keyboard-selection-p NIL)
   (:foreground-color opal:motif-gray)
   (:value (o-formula (if (gvl :selected) (gvl :string))))
   (:selected (o-formula (gvl :value)))  ;Set by interactor
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

```
(create-instance ’gg:Motif-Text-Button-Panel motif-gadget-prototype
   (:maybe-constant ’(:left :top :text-offset :final-feedback-p :toggle-p :items :font
                      :foreground-color :direction :v-spacing :h-spacing :v-align
                      :h-align :indent :fixed-width-p :fixed-width-size :fixed-height-p
                      :fixed-height-size :rank-margin :pixel-margin :visible))
   (:left 0)
   (:top 0)
   (:text-offset 5)
   (:final-feedback-p NIL)
   (:items ’("Text 1" "Text 2" "Text 3" "Text 4"))
   (:inactive-items NIL)
   (:toggle-p NIL)
   (:keyboard-selection-p NIL)
   (:keyboard-selection (o-formula ...))
   (:keyboard-selection-obj (o-formula ...))
   (:font opal:default-font)
   (:foreground-color opal:motif-gray)
   (:value-obj NIL)
   (:value (o-formula ...))
   (:active-p (o-formula ...))
   (:selection-function NIL)    ; (lambda (gadget value))
   <All customizable slots of an aggrelist>)
```

The loader file for the `motif-text-button` and `motif-text-button-panel` is "motif-text-buttons-loader".

The `motif-text-button-panel` is a set of rectangular buttons, with the string or atom associated with each button aligned inside. The button will stay depressed after the mouse is released only if `:final-feedback-p` is T.

The distance from the beginning of the longest label to the inside edge of the button frame is specified in

`:text-offset`.

4.6.2. Motif Check Buttons

```
(create-instance 'gg:Motif-Check-Button gg:motif-gadget-prototype
   (:maybe-constant '(:left :top :button-width :text-offset :text-on-left-p
                      :active-p :toggle-p :string :font :foreground-color :visible))
   (:left 0)
   (:top 0)
   (:button-width 12)
   (:text-offset 5)
   (:text-on-left-p NIL)
   (:active-p T)
   (:string "Motif Check Button")
   (:font opal:default-font)
   (:toggle-p T)
   (:keyboard-selection-p NIL)
   (:foreground-color opal:motif-gray)
   (:value (o-formula (if (gvl :selected) (gvl :string))))
   (:selected (o-formula (gvl :value))) ;Set by interactor
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

```
(create-instance 'gg:Motif-Check-Button-Panel motif-gadget-prototype
   (:maybe-constant '(:left :top :button-width :text-offset :text-on-left-p :items
                      :font :foreground-color :direction :v-spacing :h-spacing
                      :v-align :h-align :indent :fixed-width-p :fixed-width-size
                      :fixed-height-p :fixed-height-size :rank-margin :pixel-margin
                      :visible))
   (:left 0)
   (:top 0)
   (:button-width 12)
   (:text-offset 5)
   (:text-on-left-p NIL)
   (:items '("Check 1" "Check 2" "Check 3"))
   (:inactive-items NIL)
   (:keyboard-selection-p NIL)
   (:keyboard-selection (o-formula ...))
   (:keyboard-selection-obj (o-formula ...))
   (:font opal:default-font)
   (:foreground-color opal:motif-gray)
   (:value-obj NIL)
   (:value (o-formula ...))
   (:active-p (o-formula ..))
   (:selection-function NIL)    ; (lambda (gadget value))
   <All customizable slots of an aggrelist>)
```

The loader file for the `motif-check-button` and the `motif-check-button-panel` is "motif-check-buttons-loader".

The `motif-check-button-panel` is analogous to the `x-button-panel` from the standard Garnet Gadget Set. Any number of buttons may be selected at one time, and clicking on a selected button de-selects it.

Since the `motif-check-button-panel` allows selection of several items at once, the `:value` slot is a list of strings (or atoms), rather than a single string. Similarly, `:value-obj` contains a list of button objects.

The slot `:text-on-left-p` specifies whether the text will appear on the right or left of the buttons. A NIL value indicates that the text should appear on the right. When text appears on the right, the designer will probably want to set `:h-align` to `:left` in order to left-justify the text against the buttons.

The distance from the labels to the buttons is specified in `:text-offset`.

The slot `:button-width` specifies the height and width of each button square.

4.6.3. Motif Radio Buttons

```
(create-instance 'gg:Motif-Radio-Button gg:motif-gadget-prototype
   (:maybe-constant '(:left :top :button-width :text-offset :text-on-left-p
                      :toggle-p :active-p :string :font :foreground-color :visible))
   (:left 0)
   (:top 0)
   (:button-width 12)
   (:text-offset 5)
   (:text-on-left-p NIL)
   (:active-p T)
   (:string "Motif Radio Button")
   (:font opal:default-font)
   (:toggle-p T)
   (:keyboard-selection-p NIL)
   (:foreground-color opal:motif-gray)
   (:value (o-formula (if (gvl :selected) (gvl :string))))
   (:selected (o-formula (gvl :value)))  ; Set by interactor
   (:selection-function NIL)             ; (lambda (gadget value))
   )


(create-instance 'gg:Motif-Radio-Button-Panel motif-gadget-prototype
   (:maybe-constant '(:left :top :button-width :text-offset :text-on-left-p :toggle-p
                      :items :font :foreground-color :direction :v-spacing :h-spacing
                      :v-align :h-align :indent :fixed-width-p :fixed-width-size
                      :fixed-height-p :fixed-height-size :rank-margin :pixel-margin
                      :visible))
   (:left 0)
   (:top 0)
   (:button-width 12)
   (:text-offset 5)
   (:text-on-left-p NIL)
   (:items '("Radio 1" "Radio 2" "Radio 3"))
   (:inactive-items NIL)
   (:toggle-p NIL)
   (:keyboard-selection-p NIL)
   (:keyboard-selection (o-formula ...))
   (:keyboard-selection-obj (o-formula ...))
   (:font opal:default-font)
   (:foreground-color opal:motif-gray)
   (:value-obj NIL)
   (:value (o-formula ...))
   (:active-p (o-formula ...))
   (:selection-function NIL)    ; (lambda (gadget value))
   <All customizable slots of an aggrelist>)
```

The loader file for the `motif-radio-button` and `motif-radio-button-panel` is "motif-radio-buttons-loader".

The `motif-radio-button-panel` is a set of diamond buttons with items appearing to either the left or the right of the buttons (implementation of `:button-width`, `:text-on-left-p` and `:text-offset` is identical to the motif check buttons). Only one button may be selected at a time.

4.7. Motif Option Button

```
(create-instance 'gg:Motif-Option-Button opal:aggregadget
  (:maybe-constant '(:left :top :text-offset :label :button-offset :items :initial-item
                     :button-font :label-font :button-fixed-width-p :v-spacing
                     :keep-menu-in-screen-p :menu-h-align :foreground-color))
  (:left 40) (:top 40)
  (:text-offset 6)
  (:label "Option button:")
  (:button-offset 2)
  (:items '("Item 1" "Item 2" "Item 3" "Item 4"))
  (:initial-item (o-formula (first (gvl :items))))
  (:button-font opal:default-font)
  (:label-font (opal:get-standard-font NIL :bold NIL))
  (:foreground-color opal:motif-gray)
  (:value (o-formula (gvl :option-text-button :string)))
  (:button-fixed-width-p T)
  (:v-spacing 8)
  (:keep-menu-in-screen-p T)
  (:menu-h-align :left)
  (:selection-function NIL)    ; (lambda (gadget value))
  ...)
```

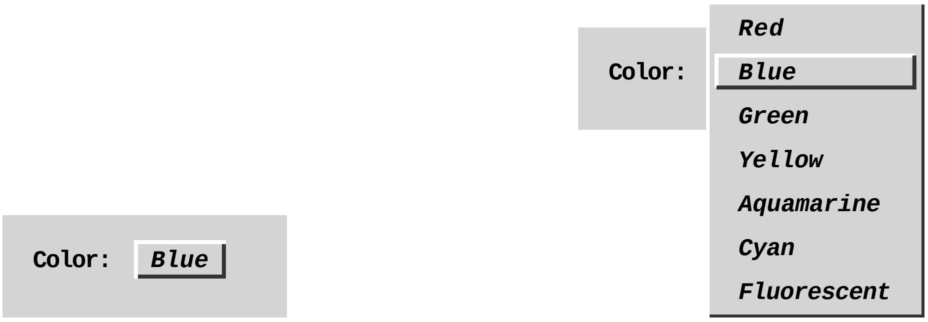


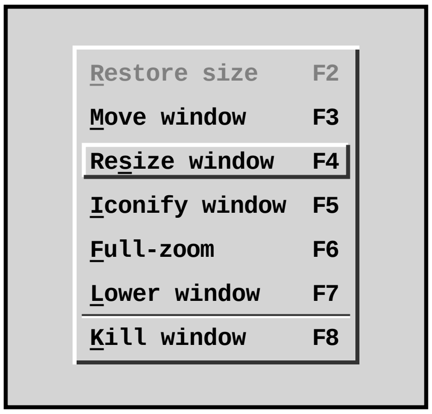
Figure 4-3: A Motif option button in its normal state (left), and showing the available options after the button is pressed (right).

This is a Motif version of the `option-button` gadget. When the left mouse button is clicked on the option button, a menu will pop up, from which items can be selected by moving the mouse over the desired item and releasing the button. The selected item will appear as the new label of the button. Figure 4-3 shows a Motif option button in its normal state and after the button has been pressed.

This button works exactly like the standard `option-button` described in section 3.6. The customizations are also alike, except that the `motif-option-button` does not have a `:button-shadow-offset` slot and adds a `:background-color` slot. The loader file for the motif option button is named "motif-option-button-loader".

4.8. Motif Menu

```
(create-instance 'gg:Motif-Menu gg:motif-gadget-prototype
   (:maybe-constant '(:left :top :min-frame-width :text-offset :v-spacing :h-align
                     :items :accelerators :bar-above-these-items :item-font
                     :accel-font :item-to-string-function :final-feedback-p
                     :foreground-color :visible))
   (:left 0)
   (:top 0)
   (:min-frame-width 0)
   (:text-offset 6)
   (:v-spacing 8)
   (:h-align :left)
   (:items '("Menu 1" "Menu 2" "Menu 3" "Menu 4" "Menu 5"))
   (:inactive-items NIL)
   (:accelerators NIL)
   (:bar-above-these-items NIL)
   (:item-to-string-function
    #'(lambda (item)
        (if item
            (if (stringp item)
                item
                (string-capitalize (string-trim ":" item)))
            "")))
   (:final-feedback-p T)
   (:keyboard-selection-p NIL)
   (:keyboard-selection (o-formula ...))
   (:keyboard-selection-obj (o-formula ...))
   (:item-font opal:default-font)
   (:accel-font opal:default-font)
   (:foreground-color opal:motif-gray)
   (:value-obj NIL)
   (:value (o-formula ...))
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```



The loader file for the `motif-menu` is "motif-menu-loader".

4.8.1. Programming Interface

The `motif-menu` is analogous to the `menu` from the standard Gadget Set, with the addition of an `:accelerators` slot which facilitates the selection of a menu item by the user. Only one item may be selected at a time.

The `:accelerators` slot is a list of triples which correspond to the items in the `:items` list. Consider the following slot definitions in an instance of the `motif-menu`:

```
(:items '("Remove-window" "Move-window" ...))
(:accelerators '((#\R "Alt+F2" :META-F2) (#\M "Alt+F3" :META-F3) ...))
```

Since the #\M character appears in the second accelerator pair, the "M" in the "Move-window" item will be underlined in the menu. The string "Alt+F3" will appear to the right of the "Move-window" item in the menu. Interactors are defined in the `motif-menu` that allow the user to press the "M" key whenever keyboard selection is activated to select "Move-window". And, after properly initializing an instance of the `motif-menu-accelerator-inter` (described below), simultaneously pressing the "Alt" and "F3" keys will also select "Move window".

Since this menu supports only single selection, the `:value` slot contains the currently selected item (from the `:items` list) and the `:value-obj` slot contains the currently selected object in the menu's aggrelist.

The `:items` and `:item-to-string-function` slots are implemented as in the `:scrolling-menu` from the standard Gadget Set (see Section 3.9). Each item (the actual item, not its string conversion) specified in the `:inactive-items` list will appear "grayed-out" and will not be selectable.

A separator bar will appear above each item listed in the slot `:bar-above-these-items`.

The minimum width of the menu frame is determined by `:min-frame-width`. The menu will appear wider than this value only if the longest item string (and its corresponding accelerator, if any) will not fit in a menu of this width.

The `:v-spacing` slot determines the distance between each item in the menu, and `:text-offset` determines the distance from the menu frame to the items (and the distance between the longest item and its corresponding accelerator, if any).

The justification of the items in the menu is determined by the slot `:h-align` and may be either `:left`, `:center`, or `:right`.

A feedback box will appear around the currently selected item if `:final-feedback-p` is T.

When the slot `:keyboard-selection-p` is T, the keyboard interface to the `motif-menu` is activated. The arrow keys will move the selection box among the items in the menu, and the return key will select the boxed item. The component of the menu's aggrelist currently surrounded by the selection box is named in `:keyboard-selection-obj`, and its string is in `:keyboard-selection`. Thus, the slot `:keyboard-selection` may be set with a string (or an atom, depending on the `:items` list) to put the selection box around an item. Since this slot contains a formula, the programmer may not supply an initial value at create-instance time. Instead, as with the `:value` slot, the user must first gv the `:keyboard-selection` slot and then s-value it to the desired initial value. **NOTE:** The return key is used to select items in the `motif-menu`, while the space-bar is used to select Motif buttons.

The fonts in which to display the items and the accelerator strings are in `:item-font` and `:accel-font`, respectively.

4.8.2. The Motif-Menu Accelerator Interactor

```
(create-instance 'gg:Motif-Menu-Accelerator-Inter inter:button-interactor
   (:window NIL)
   (:menus NIL)
   (:continuous NIL)
   (:start-where T)
   (:start-event (o-formula (multiple-value-call #'append
                              (values-list (gvl :accel-chars)))))
   (:accel-chars (o-formula (mapcar #'(lambda (menu)
                                        (gv menu :global-accel-chars))
                                    (gvl :menus))))
   (:waiting-priority gg:motif-tab-priority-level)
   (:running-priority gg:motif-tab-priority-level)
   (:stop-action #'(lambda (interactor obj-over) ...))
   (:final-function NIL))
```

The `motif-menu-accelerator-inter` interactor is used with a set of `motif-menu` instances to implement the global character selection feature (:META-F2, etc. above). When an instance is supplied with a list of menus in the `:menus` slot and the window of the menus in the `:window` slot, then when the user strikes any of the accelerator keys defined in the menus, the corresponding menu item will be selected and its functions will be executed. Only one item may be assigned to each global accelerator character. An example of the `motif-menu-accelerator-inter` may be found in demo-motif and in the `motif-menu` demo.

4.8.3. Adding Items to the Motif-Menu

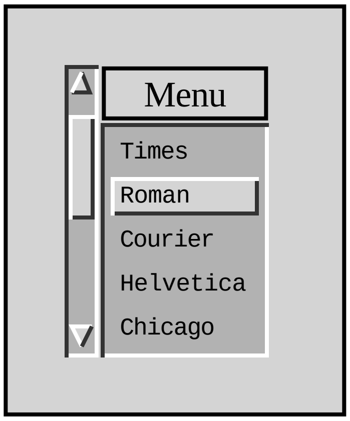
The `add-item` method for the `motif-menu` is similar to the standard method, except that the programmer may supply an accelerator to be added to the menu which corresponds to the item being added.

`opal:add-item` *motif-menu item* `[:accelerator` *accel*`] [[:where]` *position [locator]* `[:key` *function-name*`]]`

The value for *accel* should be an accelerator triplet that can be inserted into the `:accelerators` list of the *motif-menu,* such as `'(#\R "Alt+F2" :META-F2)`. Note that the accelerator parameter must come <u>before</u> the "where" keys.

The usual `remove-item` method is used for the `motif-menu`, with the additional feature that the accelerator corresponding to the old item is automatically removed from the `:accelerators` list (if there is one).

4.9. Motif Scrolling Menu



```
(create-instance 'gg:Motif-Scrolling-Menu motif-gadget-prototype
   (:maybe-constant '(:left :top :scroll-on-left-p
                      :scr-incr :page-incr :min-frame-width :v-spacing :h-align
                      :multiple-p :items :item-to-string-function
                      :item-font :num-visible :int-menu-feedback-p
                      :final-feedback-p :text-offset :title :title-font
                      :visible))
   (:left 0) (:top 0)
   (:active-p T)

   ;; Scroll bar slots
   (:scroll-on-left-p T)
   (:scr-incr 1)
   (:page-incr (o-formula (gvl :num-visible)))
   (:scroll-selection-function NIL)

   ;; Menu slots
   (:toggle-p T)
   (:min-frame-width 0)
   (:v-spacing 6)
   (:h-align :left)
   (:multiple-p T)
   (:items '("Item 1" "Item 2" "Item 3" "Item 4" "Item 5" "Item 6" "Item 7"
             "Item 8" "Item 9" "Item 10" "Item 11" "Item 12" "Item 13"
             "Item 14" "Item 15" "Item 16" "Item 17" "Item 18" "Item 19"
             "Item 20"))
   (:item-to-string-function
    #'(lambda (item)
        (if item
            (if (stringp item)
                item
                (string-capitalize (string-trim ":" item)))
            "")))
   (:item-font opal:default-font)
   (:num-visible 5)
   (:int-menu-feedback-p T)
   (:final-feedback-p T)
   (:text-offset 6)
   (:title NIL)
   (:title-font (opal:get-standard-font :serif :roman :large))
   (:menu-selection-function NIL)
   (:selected-ranks NIL)
   (:foreground-color opal:motif-gray)
   (:value (o-formula ...)))
```

The loader file for the `motif-scrolling-menu` is named "motif-scrolling-menu-loader".

The `motif-scrolling-menu` is very much like the standard `scrolling-menu`, but there are a few differences. Since the scrolling window has a motif-v-scroll-bar as a part of it, the slots `:min-scroll-bar-width`, `page-trill-p`, `:indicator-text-p`, and `:int-scroll-feedback-p` are not applicable.

Also, the `motif-scrolling-menu` has a slot `:foreground-color`, which is absent in the standard `scrolling-menu`.

4.10. Motif-Menubar

```
(create-instance 'gg:Motif-Menubar gg::motif-gadget-prototype
  (:left 0)(:top 0)
  (:items NIL)
  (:title-font opal:default-font)
  (:item-font opal:default-font)
  (:min-menubar-width 0)
  (:accelerators NIL)
  (:accelerator-windows (o-formula (gvl :window)))
  (:bar-above-these-items NIL))
```

To load the `motif-menubar`, execute `(garnet-load "gadgets:motif-menubar-loader")`.

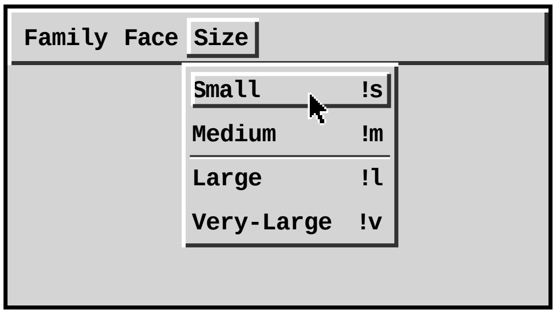


Figure 4-4: An instance of the `motif-menubar` gadget

The `motif-menubar` is used very much like the standard `menubar`, described in section 3.10. The `motif-menubar` has several additional features, including: slots that allow the menubar to extend across the top of the entire window, keyboard accelerators, and decorative "bars" in the submenus.

A simple demo which uses the `motif-menubar` is loaded along with the `motif-menubar`. To run it, execute `(gg:motif-menubar-go)`. Larger demos also use the `motif-menubar`, including `GarnetDraw` and `Demo-Multifont`.

The `:min-menubar-width` slot specifies how wide the `motif-menubar` should be. If it contains a value greater than the current width of the `motif-menubar`, the bar will extend itself. However, the items will remain fixed (i.e. they won't spread out equidistantly over the bar). This feature is useful when you want the menubar to extend across the top of the entire window, as in Figure 4-4.

4.10.1. Selection Functions

Like in the standard `menubar`, there is no `:value` slot for this gadget. The designer must use the `:selection-function` or the item functions to act on the user's selections.

There are three levels of functions in the `motif-menubar` gadget that may be called when the user makes a selection. Functions can be attached to individual submenu items, whole submenus, or the top level menubar.

All the selection functions take three parameters:

```
(lambda (gadget menu-item submenu-item))
```

The following `:items` list is taken from the `gg:Motif-Menubar-Go` demo, defined at the end of motif-menubar.lisp.

```
(:items
 '((:family ,#'family-fn
           ((:fixed ,#'fixed-fn)(:serif ,#'serif-fn)(:sans-serif ,#'sans-serif-fn)))
   (:face ,#'face-fn
          ((:roman)(:bold)(:italic)(:bold-italic)))
   (:size ,#'size-fn
          ((:small)(:medium)(:large)(:very-large)))))
```

This `:items` list attaches the functions `family-fn`, `face-fn`, and `size-fn` to each of the submenus in the menubar. Whenever the user selects an item from one of those submenus, the corresponding submenu-function will be executed.

Additionally, the functions `fixed-fn`, `serif-fn`, and `sans-serif-fn` are attached to each item in the first submenu. Whenever the user chooses "Fixed", "Serif", or "Sans-Serif" from the "Family" menu, the function associated with that item will be executed.

The order of function execution is as follows: First, the submenu-item function is called, then the submenu function, and then the top-level `:selection-function`. Notice that this is different from the order in which the functions for the regular menubar are called.

4.10.2. Accelerators

Since the `motif-menubar` uses actual instances of the `motif-menu` gadget for its submenus, the "accelerators" feature of the `motif-menu` gadget can be used in the menubar. The syntax for specifying accelerators is a bit more complicated in the menubar, because multiple submenus are used.

An accelerator is a relationship between a keyboard event and an item in the menubar. When a key is typed that corresponds to a menubar item, the function that is associated with the item is executed as if the user had pulled down the submenu and selected the item with the mouse. Each accelerator is specified by its lisp character (e.g., `:F3`), and a string to be shown to the user describing the accelerator key (e.g., "F3"). These string/character pairs are supplied to the menubar in a list, one pair for each item in the menubar. For example,

```
(:accelerators '((("!f" :|META-f|) ("!b" :|META-b|))
                 NIL
                 (NIL NIL ("!x" :|META-x|))))
```

In this accelerators list, the first item in the first submenu has accelerator string "!f", and is selected by the keyboard event, :META-f. The second item in the first submenu has the accelerator string "!b", and keyboard event :META-b. The second submenu has no accelerators. The first two items in the third submenu have no accelerators. The third item in the third submenu has string "!x" and event META-x.

In general, the format for the `:accelerators` slot is:

```
(:accelerators '(((s1,1 k1,1) (s1,2 k1,2) ...)
                 ((s2,1 k2,1) (s2,2 k2,2) ...)
                 ...))
```

where sM,N is the accelerator string for the N-th item in the M-th submenu, and kM,N is the keyboard event for the same.

The `:accelerator-windows` slot by default contains the `motif-menubar`'s window, but may contain a list of windows. When an accelerator event occurs in one of these windows, it will be perceived by the menubar and the item functions will be executed. If the mouse is in a different window, and the accelerator event occurs, the menubar will not notice the event. For this reason, you should put a list of all your application's windows in this slot, if you always want the accelerator to activate the menubar.

4.10.3. Decorative Bars

The "bars" feature of the `motif-menu` can also be used in the `motif-menubar` gadget. The `:bar-above-these-items` slot specifies over which items a horizontal line should appear. For example:

```
(:bar-above-these-items '(("Small")
                          NIL
                          ("Faster" "Warp Speed")))
```

will cause a bar to appear above the item "Small" in the first submenu, and above the items `"Faster"` and `"Warp Speed"` in the third submenu, with no bars in the second submenu. In the `motif-menubar` demo, pictured in Figure 4-4, there is a bar above third item in the last submenu.

4.10.4. Programming the Motif-Menubar the Traditional Garnet Way

There are two approaches to programming a `motif-menubar`. The first, discussed in this section, is the Garnet way, where all the `:items` are provided while creating the menubar. The second approach, discussed in section 4.10.5, requires that all the sub-objects be created individually and attached to the menubar.

The format for the `:items` slot of the `motif-menubar` is the same as in the regular `menubar`. For example,

```
(:items '(("Speed" NIL (("Slow" Slow-Fn) ("Medium" Med-Fn)
                        ("Fast" Fast-Fn) ("Whoa" Too-Fast-Fn)))))
```

This `:items` list creates a menubar with one bar-item, `"Speed"`, which has no submenu selection function. In that bar-item's submenu, are the items `"Slow"`, `"Medium"`, `"Fast"` and `"Whoa"`, which will call `Slow-Fn`, `Med-Fn`, `Fast-Fn` and `Too-Fast-Fn` respectively when selected. Note that in contrast to the example of Section 4.10.1, we did not include #' function specifiers with the selection function names. This is not necessary, because the functions are invoked with `funcall`, and the symbols will be dereferenced when necessary (though it would be faster to include the #', and avoid the dereferencing).

The submenu-items should always be described with lists, even if they have no functions (e.g., `("Slow")` instead of `"Slow"`). Also, the submenu function should either be NIL (as in the above example) or a function. As in the regular menubar, the item functions are optional and may be omitted.

4.10.4.1. An Example

The following example creates the `motif-menubar` pictured in Figure 4-4. Note the behavior of the META-f accelerator and the location of the bar.

```
(create-instance 'WIN inter:interactor-window
  (:background-color opal:motif-gray)
  (:aggregate (create-instance 'TOP-AGG opal:aggregate)))

(defun Fixed-Fn (submenu bar-item submenu-item)
  (format T "Fixed called with ~s ~s ~s~%" submenu bar-item submenu-item))

(defun Face-Fn (gadget menu-item submenu-item)
  (format T "Face called with ~s ~s ~s~%"
          gadget menu-item submenu-item))

(create-instance 'MY-MOTIF-MENUBAR gg:motif-menubar
  (:foreground-color opal:motif-gray)
  (:items
   '((:family NIL
              ((:fixed fixed-fn)(:serif)(:sans-serif)))
     (:face face-fn
            ((:roman)(:bold)(:italic)(:bold-italic)))
     (:size NIL
            ((:small)(:medium)(:large)(:very-large)))))
  (:accelerators
   '((("!f" :|META-f|) ("!e" :|META-e|) ("!a" :|META-a|))
     (("!r" :|META-r|) ("!b" :|META-b|) ("!i" :|META-i|) ("!B" :META-B))
     (("!s" :|META-s|) ("!m" :|META-m|) ("!l" :|META-l|) ("!v" :|META-v|))))
  (:bar-above-these-items
   '(NIL
     NIL
     (:large))))

(opal:add-component TOP-AGG MY-MOTIF-MENUBAR)
(opal:update WIN)
```

4.10.4.2. Adding Items to the Motif-Menubar

Adding items to the `motif-menubar` is very similar to adding items to the regular `menubar`, with the additional ability to add accelerators to the menubar along with the new items.

The add-item method for the motif-menubar may be used to add submenus:

`opal:Add-Item` *menubar submenu* `[:accelerators` *accels*`]` [*Method*]
`[[:where]` *position* `[`*locator*`] [:key` *index-function*`]]`

NOTE: If any accelerators are being added, the `:accelerators` keyword and arguments *must* appear before the `:where` arguments.

The following will add a bar item named "Volume", with a few items and accelerators in it:

```
(opal:add-item MY-MOTIF-MENUBAR
               '("Volume" NIL (("Low") ("Medium") ("High") ("Yowsa")))
               :accelerators '(NIL NIL
                                ("!h" :|META-h|) ("!y" :|META-y|))
               :before :size :key #'car)
```

To add a submenu item, use the function:

gg:Add-Submenu-Item *menubar submenu-title submenu-item* [*Method*]
  [:accelerator *accel*]
  [[:where] *position* [*locator*] [:key *index-function*]]

As with the previous function, if any accelerators are being added, they *must* appear before the `:where`. Also, notice that since only one accelerator is being added for the item, the keyword is `:accelerator`, not `:accelerators`.

The following example will add a submenu item named "Quiet" to the submenu named "Volume", and its accelerator will be META-q:

```
(gg:add-submenu-item MY-MOTIF-MENUBAR "Volume" '("Quiet")
                     :accelerator '("!q" :|META-q|) :before "Low" :key #'car)
```

4.10.4.3. Removing Items from the Motif-Menubar

An item is removed from a `motif-menubar` in exactly the same way as from a regular menubar. To remove an entire submenu, use:

opal:Remove-Item *menubar submenu* [*Method*]

For traditional Garnet programming, the *submenu* should be a sublist of the top level `:items` list, or it can just be the title of a submenu.

The following line will remove the "Volume" submenu from the previous examples.

```
(opal:remove-item MY-MOTIF-MENUBAR "Volume")
```

For removing submenu items, use

gg:Remove-Submenu-Item *menubar submenu-title submenu-item* [*Method*]

The following will remove the `:small` item from the submenu, `:size`.

```
(gg:remove-submenu-item MY-MOTIF-MENUBAR :size '(:small))
```

4.10.5. Programming the Motif-Menubar with Components

The designer may also choose a bottom-up way of programming the `motif-menubar`. The idea is to create the submenus of the menubar individually using the functions described in this section, and then attach them to a menubar.

4.10.5.1. An Example

This example creates a `motif-menubar` and several components, and attaches them together.

```
(create-instance 'WIN inter:interactor-window
  (:background-color opal:motif-blue)
  (:aggregate (create-instance 'TOP-AGG opal:aggregate)))

; Create the menubar and a bar item
(setf MY-MOTIF-MENUBAR (garnet-gadgets:make-motif-menubar))
(s-value MY-MOTIF-MENUBAR :foreground-color opal:motif-blue)

(setf MAC-BAR (garnet-gadgets:make-motif-bar-item :title "Fonts"))

; Create the submenu items
(setf SUB1 (garnet-gadgets:make-motif-submenu-item :desc '("Gothic")))
(setf SUB2 (garnet-gadgets:make-motif-submenu-item :desc '("Venice")))
(setf SUB3 (garnet-gadgets:make-motif-submenu-item :desc '("Outlaw")))

; Add submenu items to the bar item
(opal:add-item MAC-BAR SUB1)
(opal:add-item MAC-BAR SUB2)
(opal:add-item MAC-BAR SUB3 :after "Venice" :key #'car)

; Add the bar item to the menubar and update the window
(opal:add-item MY-MOTIF-MENUBAR MAC-BAR
               :accelerators '(("!g" :|META-g|) ("!v" :|META-v|) ("!o" :|META-o|)))

; Add the menubar to the top-level aggregate
(opal:add-component TOP-AGG MY-MOTIF-MENUBAR)

(opal:update WIN)
```

When programming a `motif-menubar` by components, you should add accelerators only when you add a bar-item to the menubar, or when adding a submenu item to a bar item that has already been added to a menubar. That is, you cannot add an accelerator to a submenu that has not been attached to a menubar yet.

4.10.5.2. Creating Components of the Motif-Menubar

A `motif-menubar` is essentially the same as a menubar in that there are three components - the menubar itself, bar items containing submenus, and submenu items. Each can be created with the following functions:

`gg:Make-Motif-Menubar` [*Function*]

Will return an instance of `motif-menubar`.

`gg:Make-Motif-Bar-Item &key` *desc font title* [*Function*]

Returns a `motif-bar-item`. Like the regular `menubar`, the `:desc` parameter is a description of the submenu (e.g., `'("Speed" NIL (("Fast") ("Slow") ("Crawl")))`), and the font and title keys specify the font and the heading of the submenu.

`gg:Make-Motif-Submenu-Item &key` *desc enabled* [*Function*]

Creates and returns an instance of `motif-submenu-item`, which is actually a `motif-menu-item`, since each motif-submenu is just a `motif-menu`. The `:desc` parameter describes the item, (e.g., `'("Italic")` or `'("Italic" italic-fn)`). The default for enabled is T.

4.10.5.3. Adding Components to the Motif-Menubar

Two types of components that can be added to the `motif-menubar` are bar-items and submenu-items. The `add-item` method can be used to add new bar-items to the menubar, and can also be used to add new submenu-items to existing bar-items. The `set-...` functions are used to install a collection of components all at once.

`gg:Set-Menubar` *motif-menubar new-bar-items* [*Method*]

This removes the current bar-items from *motif-menubar* and adds the new bar items. This is useful for recycling a menubar instead of creating a new one.

`gg:Set-Submenu` *motif-bar-item new-submenu-items* [*Method*]

Sets the *motif-bar-item* to have the new submenu-items. For more information on these two functions, see section 3.10.

`opal:Add-Item` *motif-menubar motif-bar-item* [`:accelerators` *accels*] [*Method*]
[[`:where`] *position* [*locator*] [`:key` *index-function*]]

Will add *motif-bar-item* to *motif-menubar*. As usual, if any accelerators are being added, the `:accelerators` key *must* be specified before the `:where` key. The `:accelerators` syntax is described in Section 4.10.2.

`opal:Add-Item` *motif-bar-item motif-menu-item* [*Method*]
[`:accelerator` *accels*]
[[`:where`] *position* [*locator*] [`:key` *index-function*]]

Adds the submenu-item, *motif-menu-item* to *motif-bar-item*. If the *motif-bar-item* is not attached to a `motif-menubar`, then no accelerators will be added, regardless of whether any are specified.

The following example shows how bar items are added to a `motif-menubar`:

```
(setf bar1 (gg:make-motif-bar-item
            :desc '("Color" NIL (("Red") ("Blue") ("Polka Dots")))))
(setf bar2 (gg:make-motif-bar-item
            :desc '("Size" NIL (("Small") ("Medium") ("Large")))))
(opal:add-item MY-MOTIF-MENUBAR bar1
               :accelerators '(("!r" :|META-r|) ("!b" :|META-b|) ("!p" :|META-p|)))
(opal:add-item MY-MOTIF-MENUBAR bar2 :before bar1)
(opal:update WIN)
```

This sequence shows how submenu-items can be attached to bar-items:

```
(setf color1 (gg:make-motif-submenu-item :desc '("Maroon")))
(setf color2 (gg:make-motif-submenu-item :desc '("Peachpuff")))
(opal:add-item bar1 color1 :accelerator '("!m" :|META-m|))
(opal:add-item bar1 color2 :after "Blue" :key #'car)
```

4.10.5.4. Removing Components from the Menubar

Bar-items and submenu-items can be removed from the menubar with the `remove-item` method.

In the example from the previous section, to remove `color1` from `bar1`, we say:

```
(opal:remove-item bar1 color1)
```

And to remove the `bar1` itself:

```
(opal:remove-item MY-MOTIF-MENUBAR bar1)
```

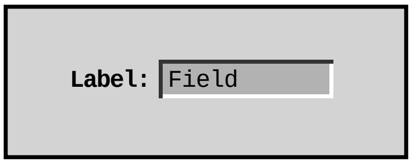
4.10.6. Methods Shared with the Regular Menubar

The following methods have the same effect on the `motif-menubar` as they have on the standard `menubar`. Please see section 3.10 for more information.

`gg:Menubar-Components` *motif-menubar* [*Method*]
`gg:Submenu-Components` *motif-bar-item* [*Method*]
`gg:Get-Bar-Component` *motif-menubar item* [*Method*]
`gg:Get-Submenu-Component` *motif-bar-item item* [*Method*]
`gg:Find-Submenu-Component` *motif-menubar submenu-title submenu-item* [*Method*]
`gg:Menubar-Disable-Component` *motif-menubar-component* [*Method*]
`gg:Menubar-Enable-Component` *motif-menubar-component* [*Method*]
`gg:Menubar-Enabled-P` *motif-menubar-component* [*Method*]
`gg:Menubar-Get-Title` *motif-menubar-component* [*Method*]
`gg:Menubar-Set-Title` *motif-menubar-component* [*Method*]
`gg:Menubar-Installed-P` *motif-menubar-component* [*Method*]

4.11. Motif-Scrolling-Labeled-Box

```
(create-instance 'gg:Motif-Scrolling-Labeled-Box motif-gadget-prototype
   (:maybe-constant '(:left :top :width :field-offset :label-offset :label-string
                      :field-font :label-font :foreground-color :active-p :visible))
   (:left 0)
   (:top 0)
   (:width 135)
   (:field-offset 4)
   (:label-offset 5)
   (:label-string "Label:")
   (:value "Field")
   (:field-font opal:default-font)     ;;**Must be fixed width**
   (:label-font (create-instance NIL opal:font (:face :bold)))
   (:foreground-color opal:motif-gray)
   (:keyboard-selection-p NIL)
   (:active-p T)
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

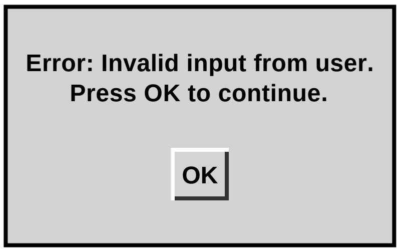


The loader file for the `motif-scrolling-labeled-box` is "motif-scrolling-labeled-box-loader".

This gadget is a Motif version of the `scrolling-labeled-box`. The `:width` of the box is fixed, and the `:value` string will scroll inside the box if it is too long to be displayed.

When the `:active-p` slot is set to NIL, both the label and the field will appear "grayed-out" and the field will not be editable.

4.12. Motif-Error-Gadget



```
(create-instance 'gg:Motif-Error-Gadget opal:aggregadget
   (:string "Error")
   (:parent-window NIL)
   (:font (opal:get-standard-font :sans-serif :bold :medium))
   (:justification :center)
   (:modal-p T)
   (:beep-p T)
   (:window NIL)          ;; Automatically initialized
   (:foreground-color opal:motif-orange)
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

The loader file for the `motif-error-gadget` is "motif-error-gadget-loader".

The `motif-error-gadget` is a dialog box that works exactly the same way as the `error-gadget` described in section 3.20. The same caveats apply, and the functions `display-error` and `display-error-and-wait` may be used to display the dialog box.

There is an additional slot provided in the `motif-error-gadget` which determines the color of the dialog box. The `:foreground-color` slot may contain any instance of `opal:color`.

4.13. Motif-Query-Gadget

```
(create-instance 'gg:Motif-Query-Gadget gg:motif-error-gadget
   (:string "Is that OK?")
   (:button-names '("OK" "CANCEL"))
   (:parent-window NIL)
   (:font (opal:get-standard-font :sans-serif :bold :medium))
   (:justification :center)
   (:modal-p T)
   (:beep-p T)
   (:window NIL)          ;; Automatically initialized
   (:foreground-color opal:motif-orange)
   (:selection-function NIL)    ; (lambda (gadget value))
   )
```

The loader file for the `motif-query-gadget` is "motif-error-gadget-loader" (it is defined in the same file as the `motif-error-gadget`).

The `motif-query-gadget` works exactly the same way as the `query-gadget` described in section 3.21. It has more buttons than the `motif-error-gadget`, so it can be used as a general-purpose dialog box. The functions `display-query` and `display-query-and-wait` may be used to display the dialog box.

The additional `:foreground-color` slot may contain any instance of `opal:color`, and determines the color of the dialog box.

4.14. Motif Save Gadget

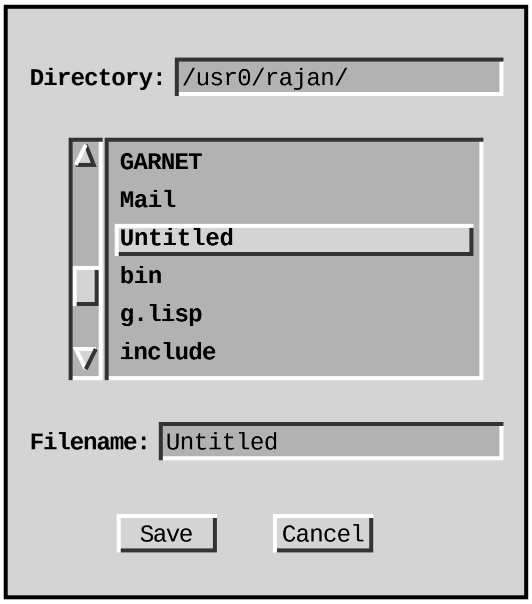


Figure 4-5: An instance of the Motif save gadget. "/usr0/rajan" is the current directory.

```
(create-instance 'motif-save-gadget opal:aggregadget
  (:maybe-constant '(:left :top :parent-window :window-title :window-left :window-top
                     :message-string :num-visible :initial-directory :button-panel-items
                     :button-panel-h-spacing :min-gadget-width :check-filenames-p
                     :modal-p :query-message :query-buttons :foreground-color
                     :dir-input-field-font :dir-input-label-font :file-input-field-font
                     :file-input-label-font :file-menu-font :button-panel-font
                     :message-font))
  (:window-title "save window")
  (:initial-directory "./")
  (:message-string "fetching directory...")
  (:num-visible 6)
  (:button-panel-items '("save" "cancel"))
  (:button-panel-h-spacing 25)
  (:min-gadget-width 240)
  (:modal-p NIL)
  (:check-filenames-p t)
  (:query-message "save over existing file")
  (:foreground-color opal:motif-light-blue)
  (:selection-function NIL)    ; (lambda (gadget value))
  (:dir-input-field-font (opal:get-standard-font NIL NIL :small))
  (:dir-input-label-font (opal:get-standard-font NIL :bold NIL))
  (:file-input-field-font (opal:get-standard-font NIL NIL :small))
  (:file-input-label-font (opal:get-standard-font NIL :bold NIL))
  (:message-font (opal:get-standard-font :fixed :italic :small))
  (:file-menu-font (opal:get-standard-font NIL :bold NIL))
  (:button-panel-font opal:default-font)
  ...)
```

The `motif-save-gadget` works exactly like the `save-gadget`, described in section 3.22. The only difference is that the `motif-save-gadget` has a slot called `:foreground-color` which allows the user to set the color of the gadget. This slot can be set to any `opal:color` object.

The loader file for the `motif-save-gadget` is named "motif-save-gadget-loader". Figure 4-5 shows an instance of the Motif save gadget.

4.15. Motif Load Gadget

```
(create-instance 'gg:Motif-Load-Gadget opal:aggregadget
  (:maybe-constant '(:left :top :parent-window :window-title :window-left
                     :window-top :dir-input-field-font :dir-input-label-font
                     :message-font :message-string :num-visible :file-menu-font
                     :initial-directory :file-input-field-font
                     :file-input-label-font :button-panel-items :button-panel-font
                     :button-panel-h-spacing :min-gadget-width :modal-p
                     :check-filenames-p :foreground-color)))
  (:parent-window NIL)
  (:window-title "load window")
  (:message-string "fetching directory...")
  (:num-visible 6)
  (:initial-directory "./")
  (:button-panel-items '("load" "cancel"))
  (:button-panel-h-spacing 25)
  (:min-gadget-width 240)
  (:modal-p NIL)
  (:check-filenames-p t)
  (:foreground-color opal:motif-light-blue)
  (:selection-function NIL)    ; (lambda (gadget value))
  (:dir-input-field-font (opal:get-standard-font NIL NIL :small))
  (:dir-input-label-font (opal:get-standard-font NIL :bold NIL))
  (:file-input-field-font (opal:get-standard-font NIL NIL :small))
  (:file-input-label-font (opal:get-standard-font NIL :bold NIL))
  (:message-font (opal:get-standard-font :fixed :italic :small))
  (:file-menu-font (opal:get-standard-font NIL :bold NIL))
  (:button-panel-font opal:default-font)
  ...)
```

The `motif-load-gadget` is loaded along with the `motif-save-gadget` by the file "motif-save-gadget-loader".

The `motif-load-gadget` works the same way as the standard `load-gadget`. The only difference is that the motif gadget has an additional `:foreground-color` slot, which can be set to any `opal:color` object.

4.16. Motif Property Sheets

The following property sheets are similar to the standard property sheets, except that they use the Motif look and feel. For a complete discussion of the use of property sheets, see section 3.24.

4.16.1. Motif-Prop-Sheet-With-OK

```
(create-instance 'gg:Motif-Prop-Sheet-With-OK opal:aggregadget
    (:maybe-constant '(:left :top :items :default-filter :ok-function :apply-function
                       :cancel-function :v-spacing :multi-line-p :select-label-p
                       :label-selected-func :label-select-event :select-value-p
                       :value-selected-func :single-select-p :foreground-color :visible))

   ; Customizable slots
    (:foreground-color opal:motif-gray) ; the color for the background
    (:left 0) (:top 0)
    (:items NIL)
    (:default-filter 'default-filter)
    (:OK-Function NIL)
    (:Apply-Function NIL)
    (:Cancel-Function NIL)
    (:v-spacing 1)
    (:pixel-margin NIL)
    (:rank-margin NIL)
    (:multi-line-p NIL)
    (:select-label-p NIL)
    (:label-select-event :any-mousedown)
    (:label-selected-func NIL)
    (:select-value-p NIL)
    (:value-selected-func NIL)
    (:single-select-p NIL)

   ; Read-only slots
    (:label-selected ...
    (:value-selected ...
    (:value ...
    (:changed-values ...
```

The loader for `motif-prop-sheet-with-OK` is "motif-prop-sheet-win-loader".

This is the same as `Prop-Sheet-With-OK` (described in section 3.24.5 except that it uses the Motif look-and-feel, and you can set the foreground color.

4.16.2. Motif-Prop-Sheet-For-Obj-With-OK

```
(create-instance 'Motif-Prop-Sheet-For-Obj-With-OK Motif-Prop-Sheet-With-OK
    (:maybe-constant '(:left :top :obj :slots :eval-p :ok-function :apply-function
                       :cancel-function :v-spacing :multi-line-p :select-label-p
                       :label-selected-func :label-select-event :select-value-p
                       :value-selected-func :single-select-p :foreground-color :visible))

   ; Customizable slots
    (:foreground-color opal:motif-gray)
    (:OK-Function NIL)
    (:Apply-Function NIL)
    (:Cancel-Function NIL)
    (:left 0) (:top 0)
    (:obj NIL)     ; a single obj or a list of objects
    (:slots NIL)   ; list of slots to show. If NIL, get from :parameters
    (:union? T)    ; if slots is NIL and multiple objects, use union or intersection of :parameters?

    (:eval-p T)    ; if T, then evaluates what the user types.  Use T for
                   ; graphical objects.  If NIL, then all the values will be strings.
    (:set-immediately-p T) ; if T then sets slots when user hits return, else doesn't
                           ; ever set the slot.
    (:type-gadgets NIL)    ; descriptor of special handling for types
    (:error-gadget NIL)    ; an error gadget to use to report errors.

    ;; plus the rest of the slots also provided by prop-sheet

    (:v-spacing 1)
    (:pixel-margin NIL)
    (:rank-margin NIL)
    (:multi-line-p NIL) ; T if multi-line strings are allowed
    (:select-label-p NIL) ; T if want to be able to select the labels
    (:label-select-event :any-mousedown)
    (:label-selected-func NIL)
    (:select-value-p NIL) ; if want to be able to select the values
    (:value-selected-func NIL)
    (:single-select-p NIL) ; to select more than one value or label

   ; Read-only slots
    (:label-selected NIL) ; set with the selected label objects (or a list)
    (:value-selected NIL) ; set with the selected value objects (or a list)
    (:value ...)   ; list of pairs of all the slots and their (filtered) values
    (:changed-values NIL)) ; only the values that have changed
```

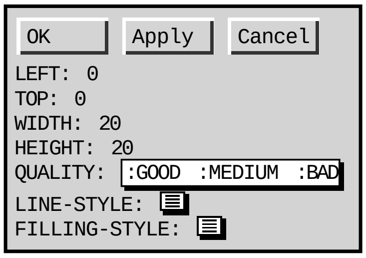


Figure 4-6: An example of `motif-prop-sheet-for-obj-with-OK` containing some gadgets. The code to create this is shown in section 3.24.9.

The loader for `motif-prop-sheet-for-obj-with-OK` is "motif-prop-sheet-win-loader".

The implementation and operation of `motif-prop-sheet-for-obj-with-ok` is identical to the `prop-sheet-for-obj-with-ok` gadget (described in section 3.24.6 with the exception that the `:foreground-color` slot may be set to any `opal:color` object.

4.17. Motif-Prop-Sheet-For-Obj-With-Done

There is a new gadget that displays a property sheet for an object and a ‘‘Done’’ button. When a slot value is edited, the slot is set immediately, rather than waiting for an OK or APPLY to be hit. Thus, the prop-sheet-for-obj slot `:set-immediately-p` is always T. This is especially useful for when the property sheet is displaying multiple objects, since slots which are not edited won’t be set. The Done button simply removes the property sheet window from the screen.

Sorry, there is no Garnet look-and-feel version of this gadget.

The parameters are pretty much the same as for `prop-sheet-for-obj`, with the addition of the `:done-function` which is called with the property sheet as a parameter.

```
(create-instance ’gg:Motif-Prop-Sheet-For-Obj-With-Done opal:aggregadget
    (:maybe-constant ’(:left :top :obj :slots :eval-p :done-function :v-spacing
                       :multi-line-p :select-label-p :label-selected-func
                       :label-select-event :select-value-p :value-selected-func
                       :single-select-p :foreground-color :visible :type-gadgets
                       :union? :error-gadget))
    (:left 5) (:top 5)
    (:obj NIL)              ; can be one object or a list of objects
    (:slots NIL)            ; list of slots to show.  If NIL uses :parameters
    (:done-function NIL)    ; called when hit done as (lambda (prop))
    (:eval-p T)             ; evaluate the values of the slots?  Usually T.
    (:error-gadget NIL)     ; used to report errors on evaluation
    (:type-gadgets NIL)     ; modifies the default display of slots
    (:union? T)   ; if slots is NIL and multiple objects, use union or intersection of :parameters?
    (:v-spacing 1)
    (:select-p NIL)         ; T if want to be able to select the entries

    (:foreground-color opal:Motif-Gray)  ; background color of the window

    (:select-label-p NIL)  ; T if want to be able to select the entries
    (:label-selected-func NIL)
    (:label-select-event :any-mousedown)
    (:select-value-p NIL)
    (:value-selected-func NIL)
    (:single-select-p NIL)

   ;; Read-Only Slots
    (:label-selected (o-formula (gvl :propsheet :label-selected)))
    (:value-selected (o-formula (gvl :propsheet :value-selected)))
    (:value (o-formula (gvl :propsheet :value)))
    (:changed-values (o-formula (gvl :propsheet :changed-values)))
    (:width (o-formula (MAX (gvl :done-panel :width)
                            (gvl :propsheet :width))))
    (:height (o-formula (+ 2 (gvl :done-panel :height)
                           (gvl :propsheet :height))))
```

The loader for `motif-prop-sheet-for-obj-with-done` is "motif-prop-sheet-win-loader".

4.18. Motif Scrolling Window

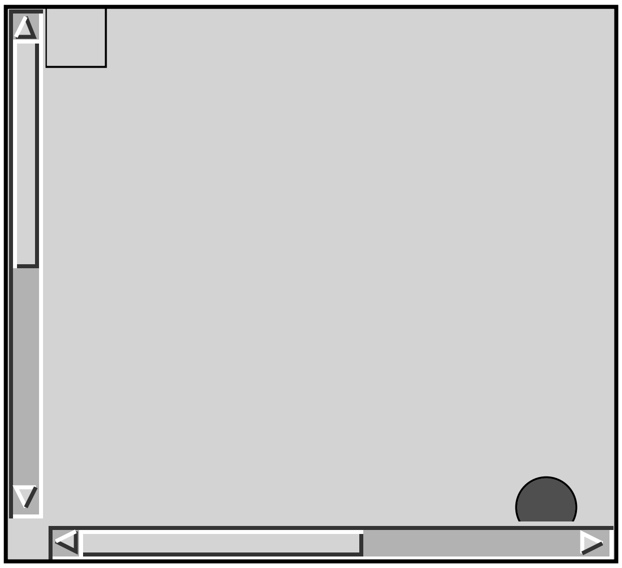
```
(create-instance 'gg:Motif-Scrolling-Window-With-Bars opal:aggregadget
   (:maybe-constant '(:left :top :width :height :border-width :title :total-width
                      :total-height :foreground-color :h-scroll-bar-p :v-scroll-bar-p
                      :h-scroll-on-top-p :v-scroll-on-left-p :h-scr-incr :h-page-incr
                      :v-scr-incr :v-page-incr :icon-title :parent-window :visible))

   ;; Window slots
   (:left 0)  ; left, top, width and height of outermost window
   (:top 0)
   (:position-by-hand NIL) ; if T, then left,top ignored and user asked for window position
   (:width 150) ;width and height of inside of outer window
   (:height 150)
   (:border-width 2) ; of outer window
   (:parent-window NIL) ; window this scrolling-window is inside of, or NIL if top level
   (:double-buffered-p NIL)
   (:omit-title-bar-p NIL)
   (:title "Motif-Scrolling-Window")
   (:icon-title (o-formula (gvl :title))) ;Default is the same as the title
   (:total-width 200)    ; total size of the scrollable area inside
   (:total-height 200)
   (:X-Offset 0)  ; x offset in of the scrollable area. CANNOT BE A FORMULA
   (:Y-Offset 0)  ; CANNOT BE A FORMULA
   (:visible T)  ; whether the window and bars are visible (mapped)
   (:foreground-color opal:motif-gray)

   (:h-scroll-bar-p T)  ; Is there a horizontal scroll bar?
   (:v-scroll-bar-p T)  ; Is there a vertical scroll bar?

   ;; Scroll Bar slots
   (:h-scroll-on-top-p NIL)  ; whether horiz scroll bar is on top or bottom
   (:v-scroll-on-left-p T)   ; whether vert scroll bar is on left or right
   (:h-scr-incr 10)  ; in pixels
   (:h-page-incr (o-formula (- (gvl :width) 10))) ; default jumps one page minus 10 pixels
   (:v-scr-incr 10)  ; in pixels
   (:v-page-incr (o-formula (- (gvl :height) 10))) ; default jumps one page minus 10 pixels

   ;; Read-Only slots
   (:Inner-Window NIL)  ; these are created by the update method
   (:inner-aggregate NIL) ; add your objects to this aggregate (but have to update first)
   (:outer-window NIL) ; call Opal:Update on this window (or on gadget itself)
   (:clip-window NIL)
   ...)
```



The loader file for the `motif-scrolling-window-with-bars` gadget is "motif-scrolling-window-loader".

The use of `motif-scrolling-window-with-bars` is identical to the `scrolling-window-with-bars` gadget described in section 3.16, with the exception that the parameters to the scroll bars are slightly different and the `:foreground-color` can be set.

Caveats:

- **If the motif-scrolling-window has a `:parent-window`, update the parent window before instantiating the motif-scrolling-window.**
- **Update the scrolling-window gadget before referring to its inner/outer windows and aggregates.**
- **The instance of the motif-scrolling-window should not be added to an aggregate.**

The `motif-scrolling-window-with-bars` gadget is not a window itself; it is an aggregadget that points to its own windows. These windows are accessed through the `:outer-window`, `:inner-window`, and `:clip-window` slots of the gadget, as in `(g-value MY-SCROLL-WIN :outer-window)`. So you cannot call `opal:make-ps-file` with the scrolling-window gadget as an argument. You have to send one of the windows that it points to:

```
> (opal:make-ps-file (g-value SCROLL-WIN :outer-window)
                     "fig.PS" :LANDSCAPE-P T :BORDERS-P :MOTIF)
T
>
```

5. Using the Gadgets: Examples

5.1. Using the :value Slot

In order to use the value returned by a gadget, we have to access the top level `:value` slot. As an example, suppose we want to make an aggregadget out of a vertical slider and a circle, and that we want the diameter of the circle to be dependent on the current value of the slider. We may create such a unit by putting a formula in the `:diameter` slot of the circle that depends on the value returned from the slider. Such an aggregadget is implemented below. The formula in the `:diameter` slot of the circle uses the KR function `gvl` to access the `:value` slot of the vertical slider.

```
(create-instance 'BALLOON opal:aggregadget
   (:parts
    `((:slider ,gg:v-slider
               (:left 10)
               (:top 20))
      (:circle ,opal:circle
               (:diameter ,(o-formula (gvl :parent :slider :value)))
               (:left 100) (:top 50)
               (:width ,(o-formula (gvl :diameter)))
               (:height ,(o-formula (gvl :diameter)))))))
```

5.2. Using the :selection-function Slot

In order to execute a function whenever any new value or item is selected (i.e., when the `:value` slot changes), that function must be specified in the slot `:selection-function`. Suppose we want a set of buttons which give us a choice of several ancient cities. We would also like to define a function which displays a message to the screen when a new city is selected. This panel can be created with the definitions below.

```
(create-instance 'MY-BUTTONS gg:text-button-panel
    (:selection-function #'Report-City-Selected)
    (:items '("Athens" "Babylon" "Rome" "Carthage")))

(defun Report-City-Selected (gadgets-object value)
  (format t "Selected city:  ~S~%~%" value)
  (format t "Pressed button object ~S~%"
            (gv gadgets-object :value-obj)))
```

5.3. Using Functions in the :items Slot

In order to execute a specific function when a specific menu item (or button) is selected, the desired function must be paired with its associated string or atom in the `:items` list. A menu which executes functions assigned to item strings appears below. Only one function (`My-Cut`) has been defined, but the definition of the others is analogous.

```
(create-instance 'MY-MENU gg:menu
   (:left 20)
   (:top 20)
   (:title "Menu")
   (:items '(("Cut" my-cut) ("Copy" my-copy) ("Paste" my-paste))))

(defun my-cut (gadgets-object item-string)
  (format t "Function CUT called~%~%"))
```

5.4. Selecting Buttons

In order to directly select a button in a button panel (rather than allowing the user to select the button with the mouse), either the `:value` or `:value-obj` slots may be set. However, neither of these slots may be given values at the time that the button panel is created (i.e., *do not supply values in the* `create-instance` *call for these slots*), since this would permanently override the formulas in the slots.

The `:value` slot may be set with any of the items (or the first element in any of the item pairs) in the `:items` list of the button panel. The example below shows how buttons on a text-button-panel and an x-button-panel could be manually selected. In both cases, the selected items (i.e., those appearing in the `:value` slot) will appear selected when the button panels are displayed in a window.

```
(create-instance 'MY-TEXT-BUTTONS gg:text-button-panel
   (:items '(:left :center :right)))
(gv MY-TEXT-BUTTONS :value)    ;; initialize the formula in the :value slot
(s-value MY-TEXT-BUTTONS :value :center)

(create-instance 'MY-X-BUTTONS gg:x-button-panel
   (:items '("Bold" "Underline" "Italic")))
(gv MY-X-BUTTONS :value)    ;; initialize the formula in the :value slot
;; Value must be a list because x-buttons have multiple selection
(s-value MY-X-BUTTONS :value '("Bold" "Underline"))
```

Buttons may also be selected by setting the `:value-obj` slot to be the actual button object or list of button objects which should be selected. This method requires the designer to look at the internal slots of the button gadgets. The example below shows how the same results may be obtained using this method as were obtained in the above example.

```
(create-instance 'NEW-TEXT-BUTTONS gg:text-buttons-panel
   (:items '(:left :center :right)))
(s-value NEW-TEXT-BUTTONS
         :value-obj
         ;; The second button corresponds to the item ":center"
         (second (gv NEW-TEXT-BUTTONS :text-button-list :components)))
```

The `:value` slot of a single button will either contain the `:string` of the button or NIL. Single buttons will appear selected when the `:value` slot contains any non-NIL value.

5.5. The :item-to-string-function Slot

The `:items` slot of the scrolling menu may be a list of any objects at all, including the standard items described in section 3.5. The mechanism which allows strings to be generated from arbitrary objects is the user-defined `:item-to-string-function`. The default scrolling menu will handle a list of standard items, but for a list of other objects a suitable function must be supplied.

As discussed in section 1.4.3, the elements of the `:items` list can be either single atoms or lists. When an element of the `:items` list is a list, then the `:item-to-string-function` is applied only to the first element in the list, rather than the list itself. In other words, the `:item-to-string-function` takes the `car` of the item list as its parameter, rather than the entire item list.

Suppose the list in the `:items` slot of the scrolling menu is

```
(list v-scroll-bar v-slider trill-device)
```

which is a list of Garnet Gadget schemas. A function must be provided which returns a string identifying an item when given a schema as input. The following slot/value pair, inserted into the definition of the new schema, will accomplish this task:

```
(:item-to-string-function #'(lambda (item)
                              (if item
                                  (name-for-schema item)  ;; imported from KR
                                  "")))
```

Index

Table of Contents