PLT Framework: GUI Application Framework

Robert Bruce Findler (robby@cs.rice.edu)
Matthew Flatt (mflatt@cs.utah.edu)

Version 202
August 2002

Copyright notice

Contents

1. This Manual

This manual describes a framework for programmers developing applications MrEd. It assumes familiarity with MrEd as described in *PLT MrEd: Graphical Toolbox Manual* and MzScheme as described in *PLT MzScheme: Language Manual*.

1.1 Thanks

Thanks to Shriram Krishnamurthi, Cormac Flanagan, Matthias Felleisen, Ian Barland, Gann Bierner, Richard Cobbe, Dan Grossman, Stephanie Weirich, Paul Steckler, Sebastian Good, Johnathan Franklin, Mark Krentel, Corky Cartwright, Michael Ernst, Kennis Koldewyn, Bruce Duba, and many others for their feedback and help.

This manual was typest using LaTeX, SLaTeX, and `tex2page`. Some typesetting macros were originally taken from Julian Smart's *Reference Manual for wxWindows 1.60: a portable C++ GUI toolkit*.

This manual was typeset on August 27, 2002.

2. Overview

The Framework is a library that provides application framework for MrEd. It is designed to make implementation of an application in MrEd simpler and easier. It provides standard classes and utilities for managing

- Frames, §11 in *PLT MrEd: Graphical Toolbox Manual*
- Editor, §7 in *PLT MrEd: Graphical Toolbox Manual*
- and many others.

See section 3.1 for information on how to load the framework into an application.

3. Preliminaries

3.1 Libraries

The framework provides these libraries:

- **Entire Framework**
  - (**require** (**lib** "framework.ss" "framework"))
    This library provides all of the definitions and syntax described in this manual.
  - (**require** (**lib** "framework-sig.ss" "framework"))
    This library provides the signature definitions: #1^, and #1^. The *framework^* signature cotains all of the names of the procedures described in this manual, except those that begin with *test:* and *gui-utils:*. The *framework-class^* signature contains all of the classes defined in this manual.
  - (**require** (**lib** "framework-unit.ss" "framework"))
    This library provides one *unit/sigs*, §**??** in *PLT MzScheme: Language Manual*: #1@. It exports the signature *framework^*. It imports the *mred^* signature.
  - (**require** (**lib** "macro.ss" "framework"))
    This defines the mixin macro. See §3.2 for more information.

- **Test Suite Engine**

  (**require** (**lib** "test.ss" "framework"))

  This library provides all of the definitions beginning with *test:* described in this manual.

- **GUI Utilities** (**require** (**lib** "gui-utils.ss" "framework"))

  This libraries provides all of the definitions beginning with *gui-utils:* described in this manual.

3.2 Mixins

The framework relies heavily on mixins. A mixin is a class parameterization modeled on a paper published by Flatt, Felleisen, and Krishnamurthi, available at `http://www.cs.rice.edu/CS/PLT/Publications/#ffk-pldi97`. The implementation of these mixins in MzScheme is with the combination of **lambda** and **class**. The framework provides a macro to simplify the checking and implementation of these mixins. It's syntax is very similar to the syntax for `class*`, §**??** in *PLT MzScheme: Language Manual*. The shape of a mixin is:

(**mixin** (*interface-expr* ...) (*interface-expr* ...)
  instance-variable-clause ...)

This macro expands into a procedure that accepts a class. The argument passed to this procedure must match the interfaces of the first *interface-expr*s expressions. The procedure returns a class that is derived from its argument. This result class must match the interfaces specified in the second *interface-expr*s section; it has clauses specified by *instance-variable-clause*s. The syntax of the *initialization-variables* and *instance-variable-clause* are exactly the same as `class*/names`, §**??** in *PLT MzScheme: Language Manual*.

The **mixin** macro does some checking to be sure that variables that the *instance-variable-clause*s refer to in their super class are in the interfaces. That checking and the checking that the input class matches the declared interfaces aside, the mixin macro's expansion is something like this:

(**mixin** (*i*<%> ...) (*j*<%> ...)
  clause ...)
=
(**lambda** (%)
  (**class*** % (*j*<%> ...)
    clause ...))

The **mixin** macro is provided by

(**require** (**lib** "macro.ss" "framework"))

3.3 GUI Test Suite Utilities

The framework provides several new primitive functions that simulate user actions, which may be used to test applications. You use these primitives and combine them just as regular MzScheme functions. For example,

```
(begin
  (test:keystroke #\A)
  (test:menu-select "File" "Save"))
```

sends a keystroke event to the window with the keyboard focus and invokes the callback function for the "Save" menu item from the "File" menu. This has the same effect as if the user typed the key "A", pulled down the "File" menu and selected "Save".

It is possible to load this portion of the framework without loading the rest of the framework. See section 3.1 for more details.

Currently, the test engine has primitives for pushing buttons, setting check-boxes and choices, sending keystrokes, selecting menu items and clicking the mouse. Many functions that are also useful in application testing, such as traversing a tree of panels, getting the text from a canvas, determining if a window is shown, and so on, exist in MrEd.

3.3.1 An example

Here is an example program that enters a factorial procedure and computes `(fact 4)`. To run this program, start DrScheme, click on the "Console" button, load this program and run `(go)`. Then bring the DrScheme window to the front and click the mouse in the DrScheme window.

```
(define go
  (lambda ()
    (sleep 3)
    (test:new-window (get-panel '(0 0 0 1)))       ; definitions canvas
    (test:menu-select "Edit" "Select All")
    (test:menu-select "Edit" "Delete")
    (type-line "(define fact")
    (type-line "(lambda (n)")
    (type-line "(if (zero? n)")
    (type-line "1")
    (type-line "(* n (fact (sub1 n))))))")
    (test:button-push (get-panel '(0 0 0 0 5 0)))  ; check-syntax button
```

```
    (test:button-push (get-panel '(0 0 0 0 5 3)))  ; execute button
    (sleep 3)
    (type-line "(fact 4)")
    (sleep 1)
    (printf "Test complete. Pending actions: ~s~n"
            (test:number-pending-actions))))

(define type-line
  (lambda (str)
    (for-each test:keystroke (string->list str))
    (test:keystroke #\return)))

(define get-panel
  (lambda (path)
    (let loop ([path  path]
               [panel (send (test:get-active-frame) get-top-panel)])
      (if (null? path)
          panel
          (loop (cdr path)
                (list-ref (ivar panel children) (car path)))))))
```

3.3.2 Actions and completeness

The actions associated with a testing primitive may not have finished when the primitive returns to its caller. Some actions may yield control before they can complete. For example, selecting "Save As..." from the "File" menu opens a dialog box and will not complete until the "OK" or "Cancel" button is pushed.

However, all testing functions wait at least a minimum interval before returning to give the action a chance to finish. This interval controls the speed at which the test suite runs, and gives some slack time for events to complete. The default interval is 100 milliseconds. The interval can be queried or set with `test:run-interval`.

A primitive action will not return until the run-interval has expired and the action has finished, raised an error, or yielded. The number of incomplete actions is given by `test:number-pending-actions`.

Note: Once a primitive action is started, it is not possible to undo it or kill its remaining effect. Thus, it is not possible to write a utility that flushes the incomplete actions and resets number-pending-actions to zero.

However, actions which do not complete right away often provide a way to cancel themselves. For example, many dialog boxes have a "Cancel" button which will terminate the action with no further effect. But this is accomplished by sending an additional action (the button push), not by undoing the original action.

3.3.3 Errors

Errors in the primitive actions (which necessarily run in the handler thread) are caught and reraised in the calling thread.

However, the primitive actions can only guarantee that the action has started, and they may return before the action has completed. As a consequence, an action may raise an error long after the function that started it has returned. In this case, the error is saved and reraised at the first opportunity (the next primitive action).

The test engine keeps a buffer for one error, saving only the first error. Any subsequent errors are discarded. Reraising an error empties the buffer, allowing the next error to be saved.

The function `test:reraise-error` reraises any pending errors.

3.3.4 Technical Issues

Active Frame

The Self Test primitive actions all implicitly apply to the top-most (active) frame.

Thread Issues

The code started by the primitive actions must run in the handler thread of the eventspace where the event takes place. As a result, the test suite that invokes the primitive actions must *not* run in that handler thread (or else some actions will deadlock). See see section 2.4 for more info.

Window Manager (Unix only)

In order for the Self Tester to work correctly, the window manager must set the keyboard focus to follow the active frame. This is the default behavior in Microsoft Windows and MacOS, but not in X windows.

In X windows, you must explicitly tell your window manager to set the keyboard focus to the top-most frame, regardless of the position of the actual mouse. Some window managers may not implement such functionality. You can obtain such an effect in Fvwm and Fvwm95 by using the option:

```
Style  "*"  ClickToFocus
```

4. Application

The procedure in this chapter is used to supply information about your application to the framework.

5. Autosave

Autosaving in MrEd is performed by registering with a single autosaver daemon. Objects that are registered with the autosaver must have a `do-autosave` method that is called periodically with no arguments.

- autosave:autosavable¡

5.1 `autosave:autosavable<%>`

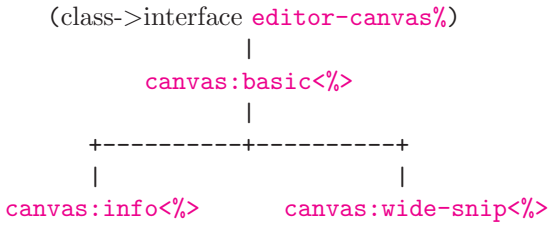
Classes that implement this interface can be autosaved.

`do-autosave`

This method is called when the object is registered to be autosaved (see `autosave:register`).

- `(`**send** *an-autosave:autosavable* `do-autosave)` ⇒ void

6. Canvas



- canvas:basic
- canvas:info
- canvas:wide-snip
- canvas:basic¡
- canvas:delegate¡
- canvas:info¡
- canvas:wide-snip¡
- canvas:basic-mixin
- canvas:delegate-mixin
- canvas:info-mixin
- canvas:wide-snip-mixin

6.1 canvas:basic<%>

Extends: (class->interface editor-canvas%)

6.2 canvas:basic-mixin

Domain: (class->interface editor-canvas%)

Implements: canvas:basic<%>

- **(instantiate** `canvas:basic-mixin%` **() (parent _) [(editor _)] [(style _)] [(scrolls-per-page _)] [(label _)] [(wheel-step _)] [(line-count _)] [(enabled _)] [(vert-margin _)] [(horiz-margin _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)])** ⇒ `canvas:basic-mixin%` object
  parent : `frame%`, `dialog%`, `panel%`, or `pane%` object
  editor = #f : `text%` or `pasteboard%` object or #f
  style = null : list of symbols in `'(no-hscroll no-vscroll hide-hscroll hide-vscroll)`
  scrolls-per-page = 100 : exact integer in [1, 10000]
  label = #f : string (up to 200 characters) or #f
  wheel-step = 3 : exact integer in [1, 10000] or #f
  line-count = #f : exact integer in [1, 1000] or #f
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  If a canvas is initialized with #f for *editor*, install an editor later with `set-editor`.

  The *style* list can contain the following flags:

  - 'no-hscroll — disallows horizontal scrolling
  - 'no-vscroll — disallows vertical scrolling
  - 'hide-hscroll — allows horizontal scrolling, but hides the horizontal scrollbar
  - 'hide-vscroll — allows vertical scrolling, but hides the vertical scrollbar

  While vertical scrolling of text editors is based on lines, horizontal scrolling and pasteboard vertical scrolling is based on a fixed number of steps per horizontal page. The *scrolls-per-page* argument sets this value.

  If provided, the *wheel-step* argument is passed on to the `wheel-step` method. The default wheel step can be overridden globally though the '|MrEd:wheelStep| preference; see "Preferences" (section 12, page 325).

  If *line-count* is not #f, it is passed on to the `set-line-count` method.

  For information about the *enabled* argument, see `window<%>`. For information about the *horiz-margin* and *vert-margin* arguments, see `subarea<%>`. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see `area<%>`.

6.3 `canvas:delegate<%>`

Extends: `canvas:basic<%>`

This class is part of the delegate window implementation.

6.4 `canvas:delegate-mixin`

Domain: `canvas:basic<%>`

Implements: `canvas:basic<%>`

Implements: canvas:delegate<%>

Provides an implementation of canvas:delegate<%>.

on-superwindow-show

Called via the event queue whenever the visibility of a window has changed, either through a call to the window's show, through the showing/hiding of one of the window's ancestors, or through the activating or deactivating of the window or its ancestor in a container (e.g., via delete-child). The method's argument indicates whether the window is now visible or not.

This method is not called when the window is initially created; it is called only after a change from the window's initial visibility. Furthermore, if a show notification event is queued for the window and it reverts its visibility before the event is dispatched, then the dispatch is canceled.

- (**send** *a-canvas:delegate-mixin* on-superwindow-show *shown?*) ⇒ void
  shown? : boolean

  Notifies the delegate window when the original window is visisble. When invisible, the blue highlighting is erased.

6.5 canvas:info<%>

Extends: canvas:basic<%>

6.6 canvas:info-mixin

Domain: canvas:basic<%>

Implements: canvas:info<%>

Implements: canvas:basic<%>

on-focus

Called when a window receives or loses the keyboard focus. If the argument is #t, the keyboard focus was received, otherwise it was lost.

Note that under X, keyboard focus can move to the menu bar when the user is selecting a menu item.

- (**send** *a-canvas:info-mixin* on-focus) ⇒ void

  Enables or disables the caret in the display's editor, if there is one.

  sets the canvas that the frame displays info about.

`set-editor`

Sets the editor that is displayed by the canvas, releasing the current editor (if any). If the new editor already has an administrator that is not associated with a `editor-canvas%`, then the new editor is *not* installed into the canvas.

- (**send** *a-canvas:info-mixin* `set-editor`) ⇒ void

  If *redraw?* is #f, then the editor is not immediately drawn; in this case, something must force a redraw later (e.g., a call to the `on-paint` method).

  If the canvas has a line count installed with `set-line-count`, the canvas's minimum height is adjusted.

  Calls `update-info` to update the frame's info panel.

6.7 canvas:wide-snip<%>

Extends: `canvas:basic<%>`

Any `canvas%` that matches this interface will automatically resize selected snips when it's size changes. Use `add-tall-snip` and `add-wide-snip` to specify which snips should be resized.

`add-tall-snip`

Snips passed to this method will be resized when the canvas's size changes. Their height will be set so they take up all of the space from their tops to the bottom of the canvas.

- (**send** *a-canvas:wide-snip* `add-tall-snip` *snip*) ⇒ void
  snip : (instance `snip%`)

`add-wide-snip`

Snips passed to this method will be resized when the canvas's size changes. Their width will be set so they take up all of the space from their lefts to the right edge of the canvas.

- (**send** *a-canvas:wide-snip* `add-wide-snip` *snip*) ⇒ void
  snip : (instance `snip%`)

`recalc-snips`

Recalculates the sizes of the wide snips.

- (**send** *a-canvas:wide-snip* `recalc-snips`) ⇒ void

6.8 canvas:wide-snip-mixin

Domain: `canvas:basic<%>`

Implements: `canvas:basic<%>`

Implements: canvas:wide-snip<%>

This canvas mantains a list of wide and tall snips and adjusts their heights and widths when the canvas's size changes.

The result of this mixin uses the same initialization arguments as the mixin's argument.

on-size

Called when the window is resized. The window's new size (in pixels) is provided to the method. The size values are for the entire window, not just the client area.

- **(send** *a-canvas:wide-snip-mixin* on-size *width height*) ⇒ void
  width : exact integer in [0, 10000]
  height : exact integer in [0, 10000]

  If the canvas is displaying an editor, its on-display-size method is called.

  Adjusts the sizes of the marked snips.

  See add-wide-snip and add-tall-snip.

6.9 canvas:basic% = (canvas:basic-mixin editor-canvas%)

canvas:basic% = (canvas:basic-mixin editor-canvas%)

- **(instantiate** canvas:basic% () **(parent** _**)** [**(editor** _**)**] [**(style** _**)**] [**(scrolls-per-page** _**)**] [**(label** _**)**] [**(wheel-step** _**)**] [**(line-count** _**)**] [**(enabled** _**)**] [**(vert-margin** _**)**] [**(horiz-margin** _**)**] [**(min-width** _**)**] [**(min-height** _**)**] [**(stretchable-width** _**)**] [**(stretchable-height** _**)**])
  ⇒ canvas:basic% object
  parent : frame%, dialog%, panel%, or pane% object
  editor = #f : text% or pasteboard% object or #f
  style = null : list of symbols in '(no-hscroll no-vscroll hide-hscroll hide-vscroll)
  scrolls-per-page = 100 : exact integer in [1, 10000]
  label = #f : string (up to 200 characters) or #f
  wheel-step = 3 : exact integer in [1, 10000] or #f
  line-count = #f : exact integer in [1, 1000] or #f
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  If a canvas is initialized with #f for *editor*, install an editor later with set-editor.

  The *style* list can contain the following flags:

  – 'no-hscroll — disallows horizontal scrolling
  – 'no-vscroll — disallows vertical scrolling
  – 'hide-hscroll — allows horizontal scrolling, but hides the horizontal scrollbar
  – 'hide-vscroll — allows vertical scrolling, but hides the vertical scrollbar

While vertical scrolling of text editors is based on lines, horizontal scrolling and pasteboard vertical scrolling is based on a fixed number of steps per horizontal page. The *scrolls-per-page* argument sets this value.

If provided, the *wheel-step* argument is passed on to the wheel-step method. The default wheel step can be overridden globally though the '|MrEd:wheelStep| preference; see "Preferences" (section 12, page 325).

If *line-count* is not #f, it is passed on to the set-line-count method.

For information about the *enabled* argument, see window<%>. For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

6.10 canvas:info% = (canvas:info-mixin canvas:basic%)

canvas:info% = (canvas:info-mixin canvas:basic%)

- **(instantiate canvas:info% () (parent _) [(editor _)] [(style _)] [(scrolls-per-page _)] [(label _)] [(wheel-step _)] [(line-count _)] [(enabled _)] [(vert-margin _)] [(horiz-margin _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)])**
  ⇒ canvas:info% object
  parent : frame%, dialog%, panel%, or pane% object
  editor = #f : text% or pasteboard% object or #f
  style = null : list of symbols in '(no-hscroll no-vscroll hide-hscroll hide-vscroll)
  scrolls-per-page = 100 : exact integer in [1, 10000]
  label = #f : string (up to 200 characters) or #f
  wheel-step = 3 : exact integer in [1, 10000] or #f
  line-count = #f : exact integer in [1, 1000] or #f
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  If a canvas is initialized with #f for *editor*, install an editor later with set-editor.

  The *style* list can contain the following flags:

  - 'no-hscroll — disallows horizontal scrolling
  - 'no-vscroll — disallows vertical scrolling
  - 'hide-hscroll — allows horizontal scrolling, but hides the horizontal scrollbar
  - 'hide-vscroll — allows vertical scrolling, but hides the vertical scrollbar

  While vertical scrolling of text editors is based on lines, horizontal scrolling and pasteboard vertical scrolling is based on a fixed number of steps per horizontal page. The *scrolls-per-page* argument sets this value.

  If provided, the *wheel-step* argument is passed on to the wheel-step method. The default wheel step can be overridden globally though the '|MrEd:wheelStep| preference; see "Preferences" (section 12, page 325).

  If *line-count* is not #f, it is passed on to the set-line-count method.

  For information about the *enabled* argument, see window<%>. For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

6.11 canvas:wide-snip% = (canvas:wide-snip-mixin canvas:basic%)

canvas:wide-snip% = (canvas:wide-snip-mixin canvas:basic%)

- **(instantiate** canvas:wide-snip% **() (parent** _**) [(editor** _**)] [(style** _**)] [(scrolls-per-page** _**)] [(label** _**)] [(wheel-step** _**)] [(line-count** _**)] [(enabled** _**)] [(vert-margin** _**)] [(horiz-margin** _**)] [(min-width** _**)] [(min-height** _**)] [(stretchable-width** _**)] [(stretchable-height** _**)])**
  ⇒ canvas:wide-snip% object
  parent : frame%, dialog%, panel%, or pane% object
  editor = #f : text% or pasteboard% object or #f
  style = null : list of symbols in '(no-hscroll no-vscroll hide-hscroll hide-vscroll)
  scrolls-per-page = 100 : exact integer in [1, 10000]
  label = #f : string (up to 200 characters) or #f
  wheel-step = 3 : exact integer in [1, 10000] or #f
  line-count = #f : exact integer in [1, 1000] or #f
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  If a canvas is initialized with #f for *editor*, install an editor later with set-editor.

  The *style* list can contain the following flags:

  - 'no-hscroll — disallows horizontal scrolling
  - 'no-vscroll — disallows vertical scrolling
  - 'hide-hscroll — allows horizontal scrolling, but hides the horizontal scrollbar
  - 'hide-vscroll — allows vertical scrolling, but hides the vertical scrollbar

  While vertical scrolling of text editors is based on lines, horizontal scrolling and pasteboard vertical scrolling is based on a fixed number of steps per horizontal page. The *scrolls-per-page* argument sets this value.

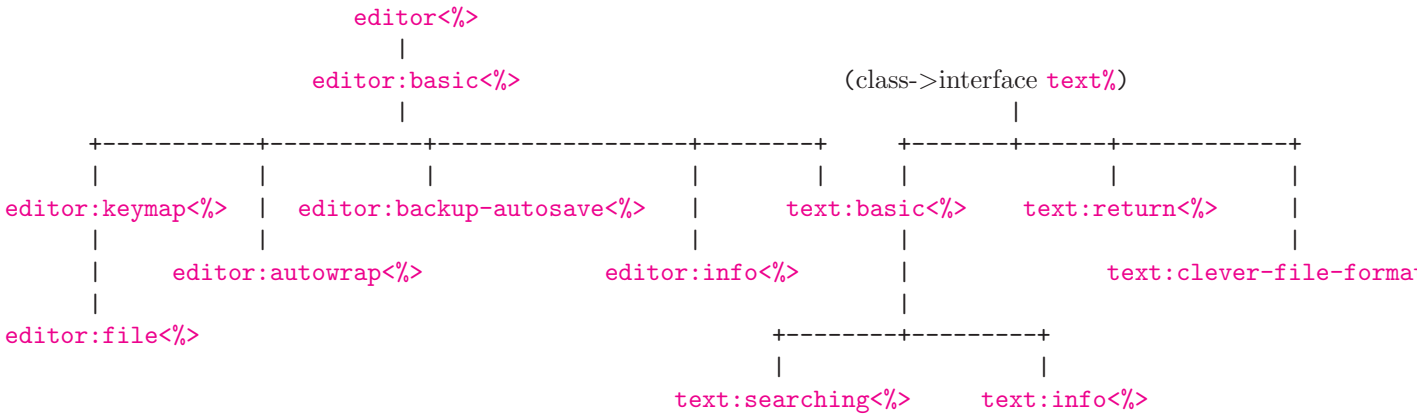
  If provided, the *wheel-step* argument is passed on to the wheel-step method. The default wheel step can be overridden globally though the '|MrEd:wheelStep| preference; see "Preferences" (section 12, page 325).

  If *line-count* is not #f, it is passed on to the set-line-count method.

  For information about the *enabled* argument, see window<%>. For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

7. Editor

This is the interface hierarchy for the editor and text classes in the framework.



- editor:autowrap¡
- editor:backup-autosave¡
- editor:basic¡
- editor:file¡
- editor:info¡
- editor:keymap¡
- editor:autowrap-mixin
- editor:backup-autosave-mixin
- editor:basic-mixin
- editor:file-mixin
- editor:info-mixin
- editor:keymap-mixin

7.1 editor:basic<%>

Extends: editor<%>

Classes matching this interface support the basic editor<%> functionality required by the framework.

`editing-this-file?`

Indicates if the file in this editor is open for editing, or merely for viewing.

- **(send** *an-editor:basic* `editing-this-file?`**)** ⇒ boolean
  Returns `#f`.

`get-top-level-window`

Returns the `top-level-window<%>` currently associated with this buffer.

This does not work for embedded editors.

- **(send** *an-editor:basic* `get-top-level-window`**)** ⇒ (union #f (implements `top-level-window<%>`))

`has-focus?`

This function returns `#t` when the editor has the keyboard focus. It is implemented using: `on-focus`

- **(send** *an-editor:basic* `has-focus?`**)** ⇒ boolean

`load-file/gui-error`

This method is an alternative to `load-file`. Rather than showing errors via the original stdout, it opens a dialog with an error message showing the error.

The result indicates if an error happened (the error has already been shown to the user). It returns `#t` if no error occurred and `#f` if an error occurred.

- **(send** *an-editor:basic* `load-file/gui-error` *filename format show-errors?*) ⇒ boolean
  filename = #f : (union string #f)
  format = 'guess : (union 'guess 'standard 'text 'text-force-cr same copy)
  show-errors? = #t : boolean

`local-edit-sequence?`

Indicates if this editor is in an edit sequence. Enclosing buffer's edit-sequence status is not considered by this method.

See `begin-edit-sequence` and `end-edit-sequence` for more info about edit sequences.

- **(send** *an-editor:basic* `local-edit-sequence?`**)** ⇒ boolean

`on-close`

This method is called when a frame that shows this buffer is closed.

- **(send** *an-editor:basic* `on-close`**)** ⇒ void
  Does nothing.

`run-after-edit-sequence`

This method is used to install callbacks that will be run after any edit-sequence completes.

- **(send** *an-editor:basic* `run-after-edit-sequence` *thunk tag***)** ⇒ void
  thunk : (-¿ void)
  tag = #f : (union symbol #f)

  The procedure *thunk* will be called immediately if the edit is not in an edit-sequence. If the edit is in an edit-sequence, it will be called when the edit-sequence completes.

  If *tag* is a symbol, the *thunk* is keyed on that symbol, and only one thunk per symbol will be called after the edit-sequence. Specifically, the last call to `run-after-edit-sequence`'s argument will be called.

`save-file-out-of-date?`

Returns `#t` if the file on disk has been modified, by some other program.

- **(send** *an-editor:basic* `save-file-out-of-date?`**)** ⇒ boolean

`save-file/gui-error`

This method is an alternative to `save-file`. Rather than showing errors via the original stdout, it opens a dialog with an error message showing the error.

The result indicates if an error happened (the error has already been shown to the user). It returns `#t` if no error occurred and `#f` if an error occurred.

- **(send** *an-editor:basic* `save-file/gui-error` *filename format show-errors?***)** ⇒ boolean
  filename = #f : (union string #f)
  format = 'same : (union 'guess 'standard 'text 'text-force-cr same copy)
  show-errors? = #t : boolean

7.2 editor:basic-mixin

Domain: `editor<%>`

Implements: `editor:basic<%>`

Implements: `editor<%>`

This provides the basic editor services required by the rest of the framework.

The result of this mixin uses the same initialization arguments as the mixin's argument.

Each instance of a class created with this mixin contains a private `keymap%` that is chained to the global keymap via: `(send` *keymap* `chain-to-keymap (keymap:get-global) #f)`.

This installs the global keymap `keymap:get-global` to handle keyboard and mouse mappings not handled by *keymap*. The global keymap is created when the framework is invoked.

`after-edit-sequence`

Called after a top-level edit sequence completes (involving unnested `begin-edit-sequence` and `end-edit-sequence`).

See also `on-edit-sequence`.

- **(send** *an-editor:basic-mixin* `after-edit-sequence`**)** ⇒ void

  Helps to implement `run-after-edit-sequence`.

`after-load-file`

Called just after the editor is loaded from a file.

The argument to the method originally specified whether the save was successful, but failures now trigger exceptions such that the method is not even called. Consequently, the argument is always #t.

See also `can-load-file?` and `on-load-file`.

- **(send** *an-editor:basic-mixin* `after-load-file` *sucess?***)** ⇒ void
  sucess? : boolean

  Updates a private instance variable with the modification time of the file, for using in implementing `save-file-out-of-date?`

`after-save-file`

Called just after the editor is saved to a file.

The argument to the method originally specified whether the save was successful, but failures now trigger exceptions such that the method is not even called. Consequently, the argument is always #t.

See also `can-save-file?` and `on-save-file`.

- **(send** *an-editor:basic-mixin* `after-save-file` *sucess?***)** ⇒ void
  sucess? : boolean

  If the current filename is not a temporary filename, this method calls `handler:add-to-recent`with the current filename.

  to add the new filename to the list of recently opened files.

  Additionally, updates a private instance variable with the modification time of the file, for using in implementing `save-file-out-of-date?`.

`can-save-file?`

Called just before the editor is saved to a file. If the return value is #f, the file is not saved. See also `on-save-file` and `after-save-file`.

- **(send** *an-editor:basic-mixin* `can-save-file?` *filename format***)** ⇒ boolean
  filename : string
  format : symbol

The *filename* argument is the name the file will be saved to. See load-file for information about *format*.

Checks to see if the file on the disk has been modified out side of this editor, using save-file-out-of-date?. If it has, this method prompts the user to be sure they want to save.

get-file

Called when the user must be queried for a filename to load an editor. A starting directory string is passed in, but is may be #f to indicate that any directory is fine.

- **(send** *an-editor:basic-mixin* get-file *directory*) ⇒ string
  directory : string or #f

  Uses finder:get-file to find a filename. Also, sets the parameter finder:dialog-parent-parameter to the result of get-top-level-window.

on-edit-sequence

Called just before a top-level (i.e., unnested) edit sequence starts.

During an edit sequence, all callbacks methods are invoked normally, but it may be appropriate for these callbacks to delay computation during an edit sequence. The callbacks must manage this delay manually. Thus, when overriding other callback methods, such as on-insert in text%, on-insert in pasteboard%, after-insert in text%, or after-insert in pasteboard%, consider overriding on-edit-sequence and after-edit-sequence as well.

"Top-level edit sequence" refers to an outermost pair of begin-edit-sequence and end-edit-sequence calls. The embedding of an editor within another editor does not affect the timing of calls to on-edit-sequence, even if the embedding editor is in an edit sequence.

- **(send** *an-editor:basic-mixin* on-edit-sequence) ⇒ boolean

  Always returns #t. Updates a flag for local-edit-sequence?

on-focus

Called when the keyboard focus changes into or out of this editor (and not to/from a snip within the editor) with #t if the focus is being turned on, #f otherwise.

- **(send** *an-editor:basic-mixin* on-focus *on?*) ⇒ void
  on? : boolean

on-new-box

Creates and returns a new snip for an embedded editor. This method is called by insert-box.

- **(send** *an-editor:basic-mixin* on-new-box *type*) ⇒ (instance editor-snip%)
  type : (union 'pasteboard 'text)

  Creates instances of pasteboard:basic% or text:basic% instead of the built in pasteboard% and text% classes.

`put-file`

Called when the user must be queried for a filename to save an editor. Starting directory and default name strings are passed in, but either may be #f to indicate that any directory is fine or there is no default name.

- **(send** *an-editor:basic-mixin* `put-file` *directory default-name*) ⇒ string
  directory : string or #f
  default-name : string or #f

  Uses `finder:put-file` to find a filename. Also, sets the parameter `finder:dialog-parent-parameter` to the result of `get-top-level-window`.

7.3 editor:keymap<%>

Extends: `editor:basic<%>`

Classes matching this interface add support for mixing in multiple keymaps. They provides an extensible interface to chained keymaps, through the `get-keymaps` method.

This editor is initialized by calling `add-editor-keymap-functions`, `add-text-keymap-functions`, and `add-pasteboard-keymap-functions`.

`get-keymaps`

The keymaps returned from this method are chained to this `editor<%>`'s keymap.

- **(send** *an-editor:keymap* `get-keymaps`) ⇒ (list-of (instance `keymap%`))

  Defaultly returns `(list keymap:get-global)`

7.4 editor:keymap-mixin

Domain: `editor:basic<%>`

Implements: `editor:basic<%>`

Implements: `editor:keymap<%>`

This provides a mixin that implements the `editor:keymap<%>` interface.

7.5 editor:autowrap<%>

Extends: `editor:basic<%>`

Classes implementing this interface keep the the `auto-wrap` state set based on the `'framework:auto-set-wrap?` preference (see section 23 for more info about preferences).

They install a preferences callback with preferences:add-callback that sets the state when the preference changes and initialize the value of auto-wrap to the current value of 'framework:auto-set-wrap? via preferences:get.

7.6 editor:autowrap-mixin

Domain: editor:basic<%>

Implements: editor:basic<%>

Implements: editor:autowrap<%>

See editor:autowrap<%>

7.7 editor:file<%>

Extends: editor:keymap<%>

Objects supporting this interface are expected to support files.

7.8 editor:file-mixin

Domain: editor:keymap<%>

Implements: editor:keymap<%>

Implements: editor:file<%>

This editor locks itself when the file that is opened is read-only in the filesystem.

The class that this mixin produces uses the same initialization arguments as it's input.

after-load-file

Called just after the editor is loaded from a file.

The argument to the method originally specified whether the save was successful, but failures now trigger exceptions such that the method is not even called. Consequently, the argument is always #t.

See also can-load-file? and on-load-file.

- **(send** *an-editor:file-mixin* after-load-file**)** ⇒ void

  Checks if the newly loaded file is write-only in the filesystem. If so, locks the editor with the lock method. Otherwise unlocks the buffer

For each canvas returned from get-canvases it checks to see if the canvas%'s get-top-level-window matches the frame:editor<%> interface. If so, it calls set-label with the last part of the filename (ie, the name of the file, not the directory the file is in).

after-save-file

Called just after the editor is saved to a file.

The argument to the method originally specified whether the save was successful, but failures now trigger exceptions such that the method is not even called. Consequently, the argument is always #t.

See also can-save-file? and on-save-file.

- **(send** *an-editor:file-mixin* after-save-file**)** ⇒ void

  Checks if the newly saved file is write-only in the filesystem. If so, locks the editor with the lock method. Otherwise unlocks the buffer

  For each canvas returned from get-canvases it checks to see if the canvas%'s get-top-level-window matches the frame:editor<%> interface. If so, it calls set-label with the last part of the filename (ie, the name of the file, not the directory the file is in).

editing-this-file?

Indicates if the file in this editor is open for editing, or merely for viewing.

- **(send** *an-editor:file-mixin* editing-this-file?**)** ⇒ boolean

  returns #t.

get-keymaps

The keymaps returned from this method are chained to this editor<%>'s keymap.

- **(send** *an-editor:file-mixin* get-keymaps**)** ⇒ (list-of (instance keymap%))

  Defaultly returns (list keymap:get-global)

  This returns a list containing the super-class's keymaps, plus the result of keymap:get-file

set-filename

Set the path name for the file to be saved from or reloaded into this editor. This method is also called when the filename changes through any method (such as load-file).

The filename of an editor can be changed by the system in response to file loading and saving method calls, and such changes do not go through this method; use on-load-file and on-save-file to monitor such filename changes.

- **(send** *an-editor:file-mixin* set-filename *name temp?***)** ⇒ void

  name : string
  temp? = #f : boolean

  Sets the filename to *filename*. If *filename* is #f or *temporary?* is a true value, then the user will still be prompted for a name on future calls to save-file and load-file.

Updates the filename on each frame displaying this editor, for each frame that matches frame:editor<%>.

7.9 editor:backup-autosave<%>

Extends: editor:basic<%>

Classes matching this interface support backup files and autosaving.

autosave?

Indicates weather this editor<%> should be autosaved.

- **(send** *an-editor:backup-autosave* autosave?) ⇒ boolean
  Returns #t.

backup?

Indicates weather this editor<%> should be backed up.

- **(send** *an-editor:backup-autosave* backup?) ⇒ boolean
  Returns the value of the preferences:get applied to 'framework:backup-files?.

do-autosave

This method is called to perform the autosaving. See also autosave:register

- **(send** *an-editor:backup-autosave* do-autosave) ⇒ void
  When the file has been modified since it was last saved and autosaving it turned on (via the autosave? method) an autosave file is created for this editor<%>.

remove-autosave

This method removes the autosave file associated with this editor<%>.

- **(send** *an-editor:backup-autosave* remove-autosave) ⇒ void

7.10 editor:backup-autosave-mixin

Domain: editor:basic<%>

Implements: autosave:autosavable<%>

Implements: editor:backup-autosave<%>

Implements: editor:basic<%>

This mixin adds backup and autosave functionality to an editor.

During initialization, this object is registered with autosave:register.

The result of this mixin uses the same initialization arguments as the mixin's argument.

on-change

Called whenever any change is made to the editor that affects the way the editor is drawn or the values reported for the location/size of some snip in the editor. The on-change method is called just before the editor calls its administrator's needs-update method to refresh the editor's display, and it is also called just before and after printing an editor.

The editor is locked for writing and reflowing during the call to on-change.

- **(send** *an-editor:backup-autosave-mixin* on-change**)** ⇒ void

  Sets a flag indicating that this editor<%> needs to be autosaved.

on-close

This method is called when a frame that shows this buffer is closed.

- **(send** *an-editor:backup-autosave-mixin* on-close**)** ⇒ void

  Does nothing.

  Deletes the autosave file and turns off autosaving.

on-save-file

Called just before the editor is saved to a file, after calling can-save-file? to verify that the save is allowed. See also after-save-file.

- **(send** *an-editor:backup-autosave-mixin* on-save-file *filename format***)** ⇒ bool
  filename : string
  format : symbol in '(guess standard text text-force-cr same copy)

  The *filename* argument is the name the file will be saved to. See load-file for information about *format*.

  If a backup file has not been created this session for this file, deletes any existing backup file and copies the old save file into the backup file. For the backup file's name, see path-utils:generate-backup-name

set-modified

Sets the modified state of the editor. Usually, the state is changed automatically after an insertion, deletion, or style change by calling this method. (This method is also called when the modification state changes through *any* method.) This method is usually not called when the state of the flag is not changing.

See also is-modified? and on-snip-modified.

- **(send** *an-editor:backup-autosave-mixin* `set-modified` *modified?*) ⇒ void
  modified? : boolean

  Sets the modification state to *modified?*. If *modified?* is true, then an internal modify-counter is set to 1.

  If *modified?* is #f and the editor's undo or redo stack contains a system-created undoer that resets the modified state (because the preceding undo or redo action puts the editor back to a state where the modification state was #f), the undoer is disabled.

  Regardless of the value of *modified?*, the editor's adminstrator's `modified` method is called.

  Finally, if *modified?* is #f and the internal modify-counter is set to 0, then the `set-unmodified` method is called on every snip within the editor.

  If the file is no longer modified, this method deletes the autosave file. If it is, it updates a flag to indicate that the autosave file is out of date.

7.11 editor:info<%>

Extends: `editor:basic<%>`

An `editor<%>` matching this interface provides information about its lock state to its `top-level-window<%>`.

7.12 editor:info-mixin

Domain: `editor:basic<%>`

Implements: `editor:basic<%>`

Implements: `editor:info<%>`

This editor tells the frame when it is locked and unlocked. See also `frame:text-info<%>`.

`lock`

Locks or unlocks the editor for modifications. If an editor is locked, *all* modifications are blocked, not just user modifications.

See also `is-locked?`.

This method does not affect internal locks, as discussed in "Locks" (section 8.8, page 169).

- **(send** *an-editor:info-mixin* `lock` *lock?*) ⇒ void
  lock? : boolean

  If *lock?* is #f, the editor is unlocked, otherwise it is locked.

  Uses `run-after-edit-sequence` to call `lock-status-changed`.

8. Exit

The exiting library provides a way to perform cleanup actions when the application is quit.

Clean up actions can be installed by registering a callback procedure that will be invoked by `exit:exit`.

On exit, *callback* will be called with no arguments. Also, use `exit:insert-on-callback`:

```
(exit:insert-on-callback callback)
```

to perform cleanup actions.

Also, callbacks can be registered that abort exiting. To install such a a callback, use `exit:insert-can?-callback`:

```
(exit:insert-can?-callback callback)
```

if *callback* returns `#f`, then the exit is aborted.

9. Exceptions

10. Finder

The procedures `finder:get-file` and `finder:put-file` query the user for a filename; `finder:get-file` gets the name of an existing file, and `finder:put-file` gets the name for a new file.

Under Windows and MacOS, `finder:get-file` and `finder:put-file` call `finder:std-get-file` and `finder:std-put-file`, which implement the filename query using platform-specific dialogs. Under Unix, `finder:get-file` and `finder:put-file` call `finder:common-get-file` and `finder:common-put-file` which use a platform-independent dialog. Which procedure `finder:get-file` and `finder:put-file` call depends on the value of the preference (see section 23 for more info about preferences) `'framework:file-dialogs`. It it is `'common`, the common platform-independent dialogs are used and if it is `'std`, the standard platform-specific dialogs are used.

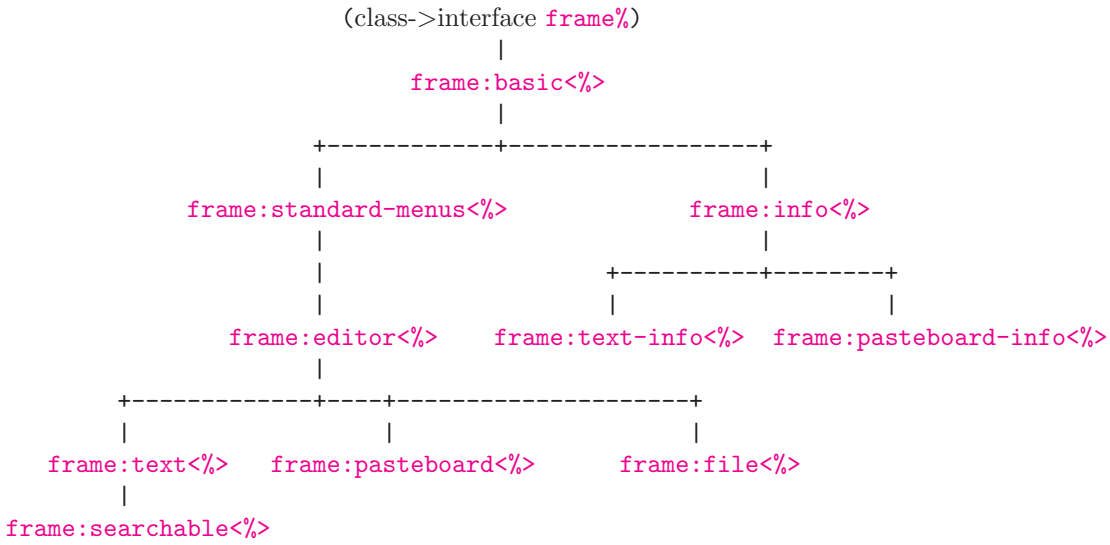
The procedure `finder:common-get-file-list` gets a list of filenames from the user using a platform-independent dialog. The arguments are the same as for `finder:get-file` and the result is either `#f` or a list of filenames.

All filenames returned by these procedures are normalized using `normalize-path` from mzlib's file utilities, section **??**.

11. Frame

This chapter describes the frame mixins, interfaces, and classes.

This is the interface hierarchy:



- frame:basic
- frame:delegate
- frame:editor
- frame:info
- frame:open-here
- frame:pasteboard-info-file
- frame:pasteboard-info
- frame:pasteboard
- frame:searchable
- frame:standard-menus
- frame:text-info-file
- frame:text-info
- frame:text

- frame:basic¡
- frame:delegate¡
- frame:editor¡
- frame:file¡
- frame:info¡
- frame:open-here¡
- frame:pasteboard-info¡
- frame:pasteboard¡
- frame:searchable-text¡
- frame:searchable¡
- frame:standard-menus¡
- frame:text-info¡
- frame:text¡
- frame:basic-mixin
- frame:delegate-mixin
- frame:editor-mixin
- frame:file-mixin
- frame:info-mixin
- frame:open-here-mixin
- frame:pasteboard-info-mixin
- frame:pasteboard-mixin
- frame:searchable-mixin
- frame:searchable-text-mixin
- frame:standard-menus-mixin
- frame:text-info-mixin
- frame:text-mixin

11.1 frame:basic<%>

Extends: `(class->interface frame%)`

Classes matching this interface support the basic `frame%` functionality required by the framework.

`close`

This method closes the frame by calling the `can-close?`, `on-close`, and `show` methods.

It's implementation is:

```
(inherit can-close? on-close)
(public
  [show
    (lambda ()
      (when (can-close?)
        (on-close)
        (show #f)))])
```

- (**send** *a-frame:basic* `close`) ⇒ void

`get-area-container`

This returns the main `area-container<%>` in the frame

- (**send** *a-frame:basic* `get-area-container`) ⇒ (instance (implements `area-container<%>`))

`get-area-container%`

The class that this method returns is used to create the `area-container<%>` in this frame.

- (**send** *a-frame:basic* `get-area-container%`) ⇒ (implements `area-container<%>`)

`get-filename`

This returns the filename that the frame is currently being saved as, or `#f` if there is no appropriate filename.

- (**send** *a-frame:basic* `get-filename` *temp*) ⇒ (union #f string)
  temp = #f : (union #f (box boolean))

  Defaultly returns `#f`.

  If *temp* is a box, it is filled with `#t` or `#f`, depending if the filename is a temporary filename.

`get-menu-bar%`

The result of this method is used to create the initial menu bar for this frame.

- (**send** *a-frame:basic* `get-menu-bar%`) ⇒ (derived-from `menu-bar%`)

  Return `menu-bar%`.

`make-root-area-container`

Override this method to insert a panel in between the panel used by the clients of this frame and the frame itself. For example, to insert a status line panel override this method with something like this:

(**class** ...

```
  ...
(rename [super-make-root-area-container make-root-area-container])
(field
  [status-panel #f])
(define/override (make-root-area-container cls parent)
  (set! status-panel
        (super-make-root-area-container vertical-panel% parent))
  (let ([root (make-object cls status-panel)])

    ; ... add other children to status-panel ...

    root))
  ...
```

In this example, status-panel will contain a root panel for the other classes, and whatever panels are needed to display status information.

The searching frame is implemented using this method.

- **(send** *a-frame:basic* `make-root-area-container` *class parent*) ⇒ (instance (implements `area-container<%>`))
  class : (implements `area-container<%>`)
  parent : (instance (implements `area-container<%>`))

  Calls `make-object` with *class* and *parent*.

11.2 frame:basic-mixin

Domain: `(class->interface frame%)`

Implements: `frame:basic<%>`

This mixin provides the basic functionality that the framework expects. It helps manage the list of frames in the `group:%` object returned by `group:get-the-frame-group`.

Do not give `panel%`s or `control<%>`s this frame as parent. Instead, use the result of the `get-area-container` method.

- **(instantiate** `frame:basic-mixin%` **()** **(label** _**)** `[`**(parent** _**)**`]` `[`**(width** _**)**`]` `[`**(height** _**)**`]` `[`**(x** _**)**`]` `[`**(y** _**)**`]` `[`**(style** _**)**`]` `[`**(enabled** _**)**`]` `[`**(border** _**)**`]` `[`**(spacing** _**)**`]` `[`**(alignment** _**)**`]` `[`**(min-width** _**)**`]` `[`**(min-height** _**)**`]` `[`**(stretchable-width** _**)**`]` `[`**(stretchable-height** _**)**`])` ⇒ `frame:basic-mixin%` object
  label : string (up to 200 characters)
  parent = #f : `frame%` object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in `'(no-resize-border no-caption no-system-menu mdi-parent mdi-child)`
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]

spacing = 0 : exact integer in [0, 1000]
alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
min-width = 0 : exact integer in [0, 10000]
min-height = 0 : exact integer in [0, 10000]
stretchable-width = #t : boolean
stretchable-height = #t : boolean

The *label* string is displayed in the frame's title bar. If the frame's label is changed (see `set-label`), the title bar is updated.

The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by `current-eventspace` . Otherwise, *parent*'s eventspace is the new frame's eventspace.

If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

The *style* flags adjust the appearance of the frame on some platforms:

- 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
- 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
- 'no-system-menu — omits the system menu (Windows)
- 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
- 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an `exn:application:mismatch` exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see `window<%>`. For information about the *border*, *spacing*, and *alignment* arguments, see `area-container<%>`. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see `area<%>`.

`after-new-child`

This method is called after a new containee area is created with this area as its container. The new child is provided as an argument to the method.

- **(send** *a-frame:basic-mixin* `after-new-child`**)** ⇒ void

  Raises an exception if attempting to add a child to this frame (except if using the `make-root-area-container` method).

`can-close?`

Called just before the window might be closed (e.g., by the window manager). If #f is returned, the window is not closed, otherwise `on-close` is called and the window is closed (i.e., the window is hidden, like calling `show` with #f).

This method is *not* called by `show` .

- **(send** *a-frame:basic-mixin* `can-close?`**)** ⇒ bool
  Calls the method `can-remove-frame?` of `group:get-the-frame-group`.

`can-exit?`

Called before `on-exit` to check whether an exit is allowed. See `on-exit` for more information.

- **(send** *a-frame:basic-mixin* `can-exit?`**)** ⇒ boolean
  Calls `exit:can-exit?`.

`on-close`

Called just before the window is closed (e.g., by the window manager). This method is *not* called by `show` .

See also `can-close?`.

- **(send** *a-frame:basic-mixin* `on-close`**)** ⇒ void
  calls the `remove-frame` method of the result of `group:get-the-frame-group`.

`on-drop-file`

For platforms that support drag-and-drop, this method is called when the user drags a file onto the window. Drag-and-drop must first be enabled for the window with `accept-drop-files`.

Under Mac OS, when the user double-clicks on a MrEd file or drags a file onto the MrEd icon, the `on-drop-file` method of the frontmost frame is called (if drag-and-drop is enabled for that frame).

- **(send** *a-frame:basic-mixin* `on-drop-file` *pathname***)** ⇒ void
  pathname : string
  Calls `handler:edit-file` with *pathname* as an argument.

`on-exit`

Called when the operating system requests that the application shut down (e.g., when the Quit menu item is selected in the main application menu under Mac OS X). In that case, this method is called for the most recently active top-level window in the initial eventspace, but only if the window's `can-exit?` method first returns true.

- **(send** *a-frame:basic-mixin* `on-exit`**)** ⇒ void
  Calls `exit:exit` with the optional argument `#f`.

`on-focus`

Called when a window receives or loses the keyboard focus. If the argument is #t, the keyboard focus was received, otherwise it was lost.

Note that under X, keyboard focus can move to the menu bar when the user is selecting a menu item.

- (**send** *a-frame:basic-mixin* `on-focus` *on?*) ⇒ void
  on? : boolean

  Does nothing.

  Calls `set-active-frame` with `this`.

`on-superwindow-show`

Called via the event queue whenever the visibility of a window has changed, either through a call to the window's `show`, through the showing/hiding of one of the window's ancestors, or through the activating or deactivating of the window or its ancestor in a container (e.g., via `delete-child`). The method's argument indicates whether the window is now visible or not.

This method is not called when the window is initially created; it is called only after a change from the window's initial visibility. Furthermore, if a show notification event is queued for the window and it reverts its visibility before the event is dispatched, then the dispatch is canceled.

- (**send** *a-frame:basic-mixin* `on-superwindow-show` *shown?*) ⇒ void
  shown? : boolean

  Notifies the result of (`group:get-the-frame-group`) that a frame has been shown, by calling the `frame-shown/hidden` method.

11.3 `frame:info<%>`

Extends: `frame:basic<%>`

Frames matching this interface support a status line.

`determine-width`

This method is used to calculate the size of an `editor-canvas%` with a particular set of characters in it. It is used to calculate the sizes of the edits in the status line.

- (**send** *a-frame:info* `determine-width` *str canvas text*) ⇒ integer
  str : string
  canvas : (instance `editor-canvas%`)
  text : (instance `text%`)

`get-info-canvas`

Returns the canvas that the `frame:info<%>` currently shows info about. See also `set-info-canvas`

- (**send** *a-frame:info* `get-info-canvas`) ⇒ (instance `canvas:basic%`)

`get-info-editor`

Override this method to specify the editor that the status line contains information about.

- (**send** *a-frame:info* `get-info-editor`) ⇒ (union #f (implements `editor<%>`))
  Returns the result of `get-editor`.

`get-info-panel`

This method returns the panel where the information about this editor is displayed.

- (**send** *a-frame:info* `get-info-panel`) ⇒ (instance `horizontal-panel%`)

`lock-status-changed`

This method is called when the lock status of the `editor<%>` changes.

- (**send** *a-frame:info* `lock-status-changed`) ⇒ void
  Updates the lock icon in the status line panel.

`set-info-canvas`

Sets this canvas to be the canvas that the info frame shows info about. The `on-focus` and `set-editor` methods call this method to ensure that the info canvas is set correctly.

- (**send** *a-frame:info* `set-info-canvas` *canvas*) ⇒ void
  canvas : (instance `canvas:basic%`)

`update-info`

This method updates all of the information in the panel.

- (**send** *a-frame:info* `update-info`) ⇒ void

11.4 `frame:info-mixin`

Domain: `frame:basic<%>`

Implements: `frame:basic<%>`

Implements: `frame:info<%>`

This mixin provides support for dispaying various info in the status line of the frame.

The result of this mixin uses the same initialization arguments as the mixin's argument.

`make-root-area-container`

Override this method to insert a panel in between the panel used by the clients of this frame and the frame itself. For example, to insert a status line panel override this method with something like this:

```
(class ...
  ...
 (rename [super-make-root-area-container make-root-area-container])
 (field
   [status-panel #f])
 (define/override (make-root-area-container cls parent)
   (set! status-panel
         (super-make-root-area-container vertical-panel% parent))
   (let ([root (make-object cls status-panel)])

     ; ... add other children to status-panel ...

     root))
  ...
```

In this example, status-panel will contain a root panel for the other classes, and whatever panels are needed to display status information.

The searching frame is implemented using this method.

- **(send** *a-frame:info-mixin* `make-root-area-container` *class parent*) ⇒ (instance `area-container<%>`)
  class : (implements `area-container<%>`)
  parent : (instance (implements `area-container<%>`))

  Calls `make-object` with *class* and *parent*.

  Builds an extra panel for displaying various information.

`on-close`

Called just before the window is closed (e.g., by the window manager). This method is *not* called by `show` .

See also `can-close?`.

- **(send** *a-frame:info-mixin* `on-close`) ⇒ void

  Removes the gc icon with `unregister-collecting-blit` and cleans up other callbacks.

11.5 `frame:text-info<%>`

Extends: `frame:info<%>`

Objects matching this interface receive information from editors constructed with `editor:info-mixin` and display it.

`anchor-status-changed`

This method is called when the anchor is turned either on or off in the `editor<%>` in this frame.

- (**send** *a-frame:text-info* anchor-status-changed) ⇒ void

anchor-status-changed

editor-position-changed

This method is called when the position in the editor<%> changes.

- (**send** *a-frame:text-info* editor-position-changed) ⇒ void

overwrite-status-changed

This method is called when the overwrite mode is turned either on or off in the editor<%> in this frame.

- (**send** *a-frame:text-info* overwrite-status-changed) ⇒ void

set-macro-recording

Shows/hides the icon in the info bar that indicates if a macro recording is in progress.

- (**send** *a-frame:text-info* set-macro-recording *on?*) ⇒ void
  on? : boolean

11.6 frame:text-info-mixin

Domain: frame:info<%>

Implements: frame:text-info<%>

Implements: frame:info<%>

This mixin adds status information to the info panel relating to an edit.

on-close

Called just before the window is closed (e.g., by the window manager). This method is *not* called by show .

See also can-close?.

- (**send** *a-frame:text-info-mixin* on-close) ⇒ void
  removes a preferences callback for 'framework:line-offsets. See section 23 for more information

update-info

This method updates all of the information in the panel.

- (**send** *a-frame:text-info-mixin* update-info) ⇒ void
  Calls overwrite-status-changed, anchor-status-changed, and editor-position-changed.

11.7 frame:pasteboard-info<%>

Extends: frame:info<%>

11.8 frame:pasteboard-info-mixin

Domain: frame:basic<%>

Implements: frame:basic<%>

Implements: frame:pasteboard-info<%>

11.9 frame:standard-menus<%>

Extends: frame:basic<%>

Classes matching this interface provides a skeleton for the standard set of menus in a frame.

The available methods, listed in order the appear in the menus, is:

- file-menu:new-callback, file-menu:create-new?, file-menu:new-string, file-menu:new-help-string, file-menu:new-on-demand, file-menu:get-new-item
- file-menu:between-new-and-open
- file-menu:open-callback, file-menu:create-open?, file-menu:open-string, file-menu:open-help-string, file-menu:open-on-demand, file-menu:get-open-item
- file-menu:open-recent-callback, file-menu:create-open-recent?, file-menu:open-recent-string, file-menu:open-recent-help-string, file-menu:open-recent-on-demand, file-menu:get-open-recent-item
- file-menu:between-open-and-revert
- file-menu:revert-callback, file-menu:create-revert?, file-menu:revert-string, file-menu:revert-help-st file-menu:revert-on-demand, file-menu:get-revert-item
- file-menu:between-revert-and-save
- file-menu:save-callback, file-menu:create-save?, file-menu:save-string, file-menu:save-help-string, file-menu:save-on-demand, file-menu:get-save-item
- file-menu:save-as-callback, file-menu:create-save-as?, file-menu:save-as-string, file-menu:save-as-hel file-menu:save-as-on-demand, file-menu:get-save-as-item
- file-menu:between-save-as-and-print
- file-menu:print-callback, file-menu:create-print?, file-menu:print-string, file-menu:print-help-string file-menu:print-on-demand, file-menu:get-print-item
- file-menu:between-print-and-close

- file-menu:close-callback, file-menu:create-close?, file-menu:close-string, file-menu:close-help-string
  file-menu:close-on-demand, file-menu:get-close-item
- file-menu:between-close-and-quit
- file-menu:quit-callback, file-menu:create-quit?, file-menu:quit-string, file-menu:quit-help-string,
  file-menu:quit-on-demand, file-menu:get-quit-item
- file-menu:after-quit
- edit-menu:undo-callback, edit-menu:create-undo?, edit-menu:undo-string, edit-menu:undo-help-string,
  edit-menu:undo-on-demand, edit-menu:get-undo-item
- edit-menu:redo-callback, edit-menu:create-redo?, edit-menu:redo-string, edit-menu:redo-help-string,
  edit-menu:redo-on-demand, edit-menu:get-redo-item
- edit-menu:between-redo-and-cut
- edit-menu:cut-callback, edit-menu:create-cut?, edit-menu:cut-string, edit-menu:cut-help-string,
  edit-menu:cut-on-demand, edit-menu:get-cut-item
- edit-menu:between-cut-and-copy
- edit-menu:copy-callback, edit-menu:create-copy?, edit-menu:copy-string, edit-menu:copy-help-string,
  edit-menu:copy-on-demand, edit-menu:get-copy-item
- edit-menu:between-copy-and-paste
- edit-menu:paste-callback, edit-menu:create-paste?, edit-menu:paste-string, edit-menu:paste-help-string
  edit-menu:paste-on-demand, edit-menu:get-paste-item
- edit-menu:between-paste-and-clear
- edit-menu:clear-callback, edit-menu:create-clear?, edit-menu:clear-string, edit-menu:clear-help-string
  edit-menu:clear-on-demand, edit-menu:get-clear-item
- edit-menu:between-clear-and-select-all
- edit-menu:select-all-callback, edit-menu:create-select-all?, edit-menu:select-all-string,
  edit-menu:select-all-help-string, edit-menu:select-all-on-demand, edit-menu:get-select-all-item
- edit-menu:between-select-all-and-find
- edit-menu:find-callback, edit-menu:create-find?, edit-menu:find-string, edit-menu:find-help-string,
  edit-menu:find-on-demand, edit-menu:get-find-item
- edit-menu:find-again-callback, edit-menu:create-find-again?, edit-menu:find-again-string,
  edit-menu:find-again-help-string, edit-menu:find-again-on-demand, edit-menu:get-find-again-item
- edit-menu:replace-and-find-again-callback, edit-menu:create-replace-and-find-again?, edit-menu:replac
  edit-menu:replace-and-find-again-help-string, edit-menu:replace-and-find-again-on-demand,
  edit-menu:get-replace-and-find-again-item
- edit-menu:between-find-and-preferences
- edit-menu:preferences-callback, edit-menu:create-preferences?, edit-menu:preferences-string,
  edit-menu:preferences-help-string, edit-menu:preferences-on-demand, edit-menu:get-preferences-item
- edit-menu:after-preferences
- help-menu:before-about
- help-menu:about-callback, help-menu:create-about?, help-menu:about-string, help-menu:about-help-string
  help-menu:about-on-demand, help-menu:get-about-item
- help-menu:after-about

`edit-menu:after-preferences`

This method is called after the addition of the preferences menu-item to the edit-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `edit-menu:after-preferences` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Does nothing.

`edit-menu:between-clear-and-select-all`

This method is called between the addition of the clear menu-item and before the addition of the select-all menu-item to the edit-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `edit-menu:between-clear-and-select-all` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Does nothing.

`edit-menu:between-copy-and-paste`

This method is called between the addition of the copy menu-item and before the addition of the paste menu-item to the edit-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `edit-menu:between-copy-and-paste` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Does nothing.

`edit-menu:between-cut-and-copy`

This method is called between the addition of the cut menu-item and before the addition of the copy menu-item to the edit-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `edit-menu:between-cut-and-copy` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Does nothing.

`edit-menu:between-find-and-preferences`

This method is called between the addition of the find menu-item and before the addition of the preferences menu-item to the edit-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `edit-menu:between-find-and-preferences` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Adds a separator except when `current-eventspace-has-standard-menus?` returns `#t`.

`edit-menu:between-paste-and-clear`

This method is called between the addition of the paste menu-item and before the addition of the clear menu-item to the edit-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* edit-menu:between-paste-and-clear *menu*) ⇒ void
  menu : (instance (derived-from menu%))

  Does nothing.

edit-menu:between-redo-and-cut

This method is called between the addition of the redo menu-item and before the addition of the cut menu-item to the edit-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* edit-menu:between-redo-and-cut *menu*) ⇒ void
  menu : (instance (derived-from menu%))

  Adds a separator menu item.

edit-menu:between-select-all-and-find

This method is called between the addition of the select-all menu-item and before the addition of the find menu-item to the edit-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* edit-menu:between-select-all-and-find *menu*) ⇒ void
  menu : (instance (derived-from menu%))

  Adds a separator menu item.

edit-menu:clear-callback

This method is called when the clear menu-item of the edit-menu menu is selected.

- (**send** *a-frame:standard-menus* edit-menu:clear-callback *item evt*) ⇒ void
  item : (instance (derived-from menu-item%))
  evt : (instance control-event%)

  Defaultly bound to: (**lambda** (*menu evt*) (**let** ((*edit* (*get-edit-target-object*))) (**when** (**and** *edit* (is-a? *edit editor<%>*)) (**send** *edit do-edit-operation* (**quote** clear)))) #t)

edit-menu:clear-help-string

This result of this method is used as the help string when the menu-item% object is created.

- (**send** *a-frame:standard-menus* edit-menu:clear-help-string) ⇒ string

  Defaultly returns "(*string-constant clear-info*)"

edit-menu:clear-on-demand

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* edit-menu:clear-on-demand *item*) ⇒ void
  item : menu-item%

  Defaultly is this: (**lambda** (*item*) (**let*** ((*editor* (*get-edit-target-object*)) (*enable?* (**and** *editor* (is-a? *editor editor<\%>*) (**send** *editor can-do-edit-operation?* (**quote** clear))))) (**send** *item enable enable?*)))

`edit-menu:clear-string`

The result of this method is the name of this menu.

- **(send** *a-frame:standard-menus* `edit-menu:clear-string`**)** ⇒ string

  defaultly returns "(if (eq? (system-type) (quote windows)) (string-constant clear-menu-item-windows) (string-constant clear-menu-item-windows))"

`edit-menu:copy-callback`

This method is called when the copy menu-item of the edit-menu menu is selected.

- **(send** *a-frame:standard-menus* `edit-menu:copy-callback` *item evt***)** ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)

  Defaultly bound to: (**lambda** (*menu evt*) (**let** ((*edit* (*get-edit-target-object*))) (**when** (**and** *edit* (`is-a?` *edit editor<%>*)) (**send** *edit do-edit-operation* (**quote** copy)))) #t)

`edit-menu:copy-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- **(send** *a-frame:standard-menus* `edit-menu:copy-help-string`**)** ⇒ string

  Defaultly returns "(*string-constant copy-info*)"

`edit-menu:copy-on-demand`

The menu item's on-demand method calls this method

- **(send** *a-frame:standard-menus* `edit-menu:copy-on-demand` *item***)** ⇒ void
  item : `menu-item%`

  Defaultly is this: (**lambda** (*item*) (**let*** ((*editor* (*get-edit-target-object*)) (*enable?* (**and** *editor* (`is-a?` *editor editor<%>*) (**send** *editor can-do-edit-operation?* (**quote** copy))))) (**send** *item enable enable?*)))

`edit-menu:copy-string`

The result of this method is the name of this menu.

- **(send** *a-frame:standard-menus* `edit-menu:copy-string`**)** ⇒ string

  defaultly returns "(string-constant copy-menu-item)"

`edit-menu:create-clear?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `edit-menu:create-clear?`**)** ⇒ `boolean`

  defaultly returns (lambda (menu evt) (let ((edit (get-edit-target-object))) (when (and edit (is-a? edit editor¡%¿)) (send edit do-edit-operation (quote clear)))) #t)

`edit-menu:create-copy?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `edit-menu:create-copy?`**)** ⇒ `boolean`

  defaultly returns (lambda (menu evt) (let ((edit (get-edit-target-object))) (when (and edit (is-a? edit editor¡%¿)) (send edit do-edit-operation (quote copy)))) #t)

`edit-menu:create-cut?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `edit-menu:create-cut?`**)** ⇒ `boolean`

  defaultly returns (lambda (menu evt) (let ((edit (get-edit-target-object))) (when (and edit (is-a? edit editor¡%¿)) (send edit do-edit-operation (quote cut)))) #t)

`edit-menu:create-find-again?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `edit-menu:create-find-again?`**)** ⇒ `boolean`

  defaultly returns (lambda (item control) (void))

`edit-menu:create-find?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `edit-menu:create-find?`**)** ⇒ `boolean`

  defaultly returns (lambda (item control) (void))

`edit-menu:create-paste?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `edit-menu:create-paste?`**)** ⇒ `boolean`

  defaultly returns (lambda (menu evt) (let ((edit (get-edit-target-object))) (when (and edit (is-a? edit editor¡%¿)) (send edit do-edit-operation (quote paste)))) #t)

`edit-menu:create-preferences?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `edit-menu:create-preferences?`**)** ⇒ `boolean`

  defaultly returns (lambda (item control) (preferences:show-dialog) #t)

`edit-menu:create-redo?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `edit-menu:create-redo?`**)** ⇒ `boolean`

  defaultly returns (lambda (menu evt) (let ((edit (get-edit-target-object))) (when (and edit (is-a? edit editor¡%¿)) (send edit do-edit-operation (quote redo)))) #t)

`edit-menu:create-replace-and-find-again?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `edit-menu:create-replace-and-find-again?`**)** ⇒ `boolean`

  defaultly returns (lambda (item control) (void))

`edit-menu:create-select-all?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `edit-menu:create-select-all?`**)** ⇒ `boolean`

  defaultly returns (lambda (menu evt) (let ((edit (get-edit-target-object))) (when (and edit (is-a? edit editor¡%¿)) (send edit do-edit-operation (quote select-all)))) #t)

`edit-menu:create-undo?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `edit-menu:create-undo?`**)** ⇒ `boolean`

  defaultly returns (lambda (menu evt) (let ((edit (get-edit-target-object))) (when (and edit (is-a? edit editor¡%¿)) (send edit do-edit-operation (quote undo)))) #t)

`edit-menu:cut-callback`

This method is called when the cut menu-item of the edit-menu menu is selected.

- (**send** *a-frame:standard-menus* edit-menu:cut-callback *item evt*) ⇒ void
  item : (instance (derived-from menu-item%))
  evt : (instance control-event%)

  Defaultly bound to: (**lambda** (*menu evt*) (**let** ((*edit* (*get-edit-target-object*))) (**when** (**and** *edit* (is-a? *edit editor<%>*)) (**send** *edit do-edit-operation* (**quote** cut)))) #t)

edit-menu:cut-help-string

This result of this method is used as the help string when the menu-item% object is created.

- (**send** *a-frame:standard-menus* edit-menu:cut-help-string) ⇒ string

  Defaultly returns "(*string-constant cut-info*)"

edit-menu:cut-on-demand

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* edit-menu:cut-on-demand *item*) ⇒ void
  item : menu-item%

  Defaultly is this: (**lambda** (*item*) (**let*** ((*editor* (*get-edit-target-object*)) (*enable?* (**and** *editor* (is-a? *editor editor<\%>*) (**send** *editor can-do-edit-operation?* (**quote** cut))))) (**send** *item enable enable?*)))

edit-menu:cut-string

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* edit-menu:cut-string) ⇒ string

  defaultly returns "(string-constant cut-menu-item)"

edit-menu:find-again-callback

This method is called when the find-again menu-item of the edit-menu menu is selected.

- (**send** *a-frame:standard-menus* edit-menu:find-again-callback *item evt*) ⇒ void
  item : (instance (derived-from menu-item%))
  evt : (instance control-event%)

  Defaultly bound to: (**lambda** (*item control*) (void))

edit-menu:find-again-help-string

This result of this method is used as the help string when the menu-item% object is created.

- (**send** *a-frame:standard-menus* edit-menu:find-again-help-string) ⇒ string

  Defaultly returns "(*string-constant find-again-info*)"

`edit-menu:find-again-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `edit-menu:find-again-on-demand` *item*) ⇒ void
  item : `menu-item%`
  Defaultly is this: (**lambda** (*item*) (**send** *item enable* (**let** ((*target* (*get-edit-target-object*))) (**and** *target* (`is-a?` *target editor<\%>*)))))

`edit-menu:find-again-string`

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* `edit-menu:find-again-string`) ⇒ string
  defaultly returns "(string-constant find-again-menu-item)"

`edit-menu:find-callback`

This method is called when the find menu-item of the edit-menu menu is selected.

- (**send** *a-frame:standard-menus* `edit-menu:find-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Defaultly bound to: (**lambda** (*item control*) (`void`))

`edit-menu:find-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- (**send** *a-frame:standard-menus* `edit-menu:find-help-string`) ⇒ string
  Defaultly returns "(*string-constant find-info*)"

`edit-menu:find-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `edit-menu:find-on-demand` *item*) ⇒ void
  item : `menu-item%`
  Defaultly is this: (**lambda** (*item*) (**send** *item enable* (**let** ((*target* (*get-edit-target-object*))) (**and** *target* (`is-a?` *target editor<\%>*)))))

`edit-menu:find-string`

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* `edit-menu:find-string`) ⇒ string
  defaultly returns "(string-constant find-menu-item)"

`edit-menu:get-clear-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `edit-menu:get-clear-item`) ⇒ (instance `menu-item%`)

`edit-menu:get-copy-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `edit-menu:get-copy-item`) ⇒ (instance `menu-item%`)

`edit-menu:get-cut-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `edit-menu:get-cut-item`) ⇒ (instance `menu-item%`)

`edit-menu:get-find-again-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `edit-menu:get-find-again-item`) ⇒ (instance `menu-item%`)

`edit-menu:get-find-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `edit-menu:get-find-item`) ⇒ (instance `menu-item%`)

`edit-menu:get-paste-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `edit-menu:get-paste-item`) ⇒ (instance `menu-item%`)

`edit-menu:get-preferences-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `edit-menu:get-preferences-item`) ⇒ (instance `menu-item%`)

`edit-menu:get-redo-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `edit-menu:get-redo-item`) ⇒ (instance `menu-item%`)

`edit-menu:get-replace-and-find-again-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `edit-menu:get-replace-and-find-again-item`) ⇒ (instance `menu-item%`)

`edit-menu:get-select-all-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `edit-menu:get-select-all-item`) ⇒ (instance `menu-item%`)

`edit-menu:get-undo-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `edit-menu:get-undo-item`) ⇒ (instance `menu-item%`)

`edit-menu:paste-callback`

This method is called when the paste menu-item of the edit-menu menu is selected.

- (**send** *a-frame:standard-menus* `edit-menu:paste-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Defaultly bound to: (**lambda** (*menu evt*) (**let** ((*edit* (*get-edit-target-object*))) (**when** (**and** *edit* (`is-a?` *edit editor<%>*)) (**send** *edit do-edit-operation* (**quote** paste)))) #t)

`edit-menu:paste-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- (**send** *a-frame:standard-menus* `edit-menu:paste-help-string`) ⇒ string
  Defaultly returns "(*string-constant paste-info*)"

`edit-menu:paste-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `edit-menu:paste-on-demand` *item*) ⇒ void
  item : `menu-item%`
  Defaultly is this: (**lambda** (*item*) (**let*** ((*editor* (*get-edit-target-object*)) (*enable?* (**and** *editor* (`is-a?` *editor editor<%>*) (**send** *editor can-do-edit-operation?* (**quote** paste))))) (**send** *item enable enable?*)))

`edit-menu:paste-string`

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* edit-menu:paste-string) ⇒ string
  defaultly returns "(string-constant paste-menu-item)"

edit-menu:preferences-callback

This method is called when the preferences menu-item of the edit-menu menu is selected.

- (**send** *a-frame:standard-menus* edit-menu:preferences-callback *item evt*) ⇒ void
  item : (instance (derived-from menu-item%))
  evt : (instance control-event%)
  Defaultly bound to: (**lambda** (*item control*) (*preferences:show-dialog*) #t)

edit-menu:preferences-help-string

This result of this method is used as the help string when the menu-item% object is created.

- (**send** *a-frame:standard-menus* edit-menu:preferences-help-string) ⇒ string
  Defaultly returns "(*string-constant preferences-info*)"

edit-menu:preferences-on-demand

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* edit-menu:preferences-on-demand *item*) ⇒ void
  item : menu-item%
  Defaultly is this: (**lambda** (*menu-item*) (void))

edit-menu:preferences-string

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* edit-menu:preferences-string) ⇒ string
  defaultly returns "(string-constant preferences-menu-item)"

edit-menu:redo-callback

This method is called when the redo menu-item of the edit-menu menu is selected.

- (**send** *a-frame:standard-menus* edit-menu:redo-callback *item evt*) ⇒ void
  item : (instance (derived-from menu-item%))
  evt : (instance control-event%)
  Defaultly bound to: (**lambda** (*menu evt*) (**let** ((*edit* (*get-edit-target-object*))) (**when** (**and** *edit* (is-a? *edit editor<%>*)) (**send** *edit do-edit-operation* (**quote** redo)))) #t)

edit-menu:redo-help-string

This result of this method is used as the help string when the menu-item% object is created.

- (**send** *a-frame:standard-menus* `edit-menu:redo-help-string`) ⇒ string

  Defaultly returns "(*string-constant redo-info*)"

`edit-menu:redo-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `edit-menu:redo-on-demand` *item*) ⇒ void
  item : `menu-item%`

  Defaultly is this: (**lambda** (*item*) (**let*** ((*editor* (*get-edit-target-object*)) (*enable?* (**and** *editor* (`is-a?` *editor editor<\%>*) (**send** *editor can-do-edit-operation?* (**quote** redo))))) (**send** *item enable enable?*)))

`edit-menu:redo-string`

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* `edit-menu:redo-string`) ⇒ string

  defaultly returns "(string-constant redo-menu-item)"

`edit-menu:replace-and-find-again-callback`

This method is called when the replace-and-find-again menu-item of the edit-menu menu is selected.

- (**send** *a-frame:standard-menus* `edit-menu:replace-and-find-again-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)

  Defaultly bound to: (**lambda** (*item control*) (`void`))

`edit-menu:replace-and-find-again-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- (**send** *a-frame:standard-menus* `edit-menu:replace-and-find-again-help-string`) ⇒ string

  Defaultly returns "(*string-constant replace-and-find-again-info*)"

`edit-menu:replace-and-find-again-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `edit-menu:replace-and-find-again-on-demand` *item*) ⇒ void
  item : `menu-item%`

  Defaultly is this: (**lambda** (*item*) (**send** *item enable* (**let** ((*target* (*get-edit-target-object*))) (**and** *target* (`is-a?` *target editor<\%>*)))))

edit-menu:replace-and-find-again-string

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* edit-menu:replace-and-find-again-string) ⇒ string
  defaultly returns "(string-constant replace-and-find-again-menu-item)"

edit-menu:select-all-callback

This method is called when the select-all menu-item of the edit-menu menu is selected.

- (**send** *a-frame:standard-menus* edit-menu:select-all-callback *item evt*) ⇒ void
  item : (instance (derived-from menu-item%))
  evt : (instance control-event%)
  Defaultly bound to: (**lambda** (*menu evt*) (**let** ((*edit* (*get-edit-target-object*))) (**when** (**and** *edit* (is-a? *edit editor<%>*)) (**send** *edit do-edit-operation* (**quote** select-all)))) #t)

edit-menu:select-all-help-string

This result of this method is used as the help string when the menu-item% object is created.

- (**send** *a-frame:standard-menus* edit-menu:select-all-help-string) ⇒ string
  Defaultly returns "(*string-constant select-all-info*)"

edit-menu:select-all-on-demand

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* edit-menu:select-all-on-demand *item*) ⇒ void
  item : menu-item%
  Defaultly is this: (**lambda** (*item*) (**let*** ((*editor* (*get-edit-target-object*)) (*enable?* (**and** *editor* (is-a? *editor editor<\%>*) (**send** *editor can-do-edit-operation?* (**quote** select-all))))) (**send** *item enable enable?*)))

edit-menu:select-all-string

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* edit-menu:select-all-string) ⇒ string
  defaultly returns "(string-constant select-all-menu-item)"

edit-menu:undo-callback

This method is called when the undo menu-item of the edit-menu menu is selected.

- (**send** *a-frame:standard-menus* edit-menu:undo-callback *item evt*) ⇒ void
  item : (instance (derived-from menu-item%))
  evt : (instance control-event%)

Defaultly bound to: (**lambda** (*menu evt*) (**let** ((*edit* (*get-edit-target-object*))) (**when** (**and** *edit* (`is-a?` *edit editor<%>*)) (**send** *edit do-edit-operation* (**quote** undo)))) #t)

`edit-menu:undo-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- (**send** *a-frame:standard-menus* `edit-menu:undo-help-string`) ⇒ string

  Defaultly returns "(*string-constant undo-info*)"

`edit-menu:undo-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `edit-menu:undo-on-demand` *item*) ⇒ void
  item : `menu-item%`

  Defaultly is this: (**lambda** (*item*) (**let*** ((*editor* (*get-edit-target-object*)) (*enable?* (**and** *editor* (`is-a?` *editor editor<%>*) (**send** *editor can-do-edit-operation?* (**quote** undo))))) (**send** *item enable enable?*)))

`edit-menu:undo-string`

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* `edit-menu:undo-string`) ⇒ string

  defaultly returns "(string-constant undo-menu-item)"

`file-menu:after-quit`

This method is called after the addition of the quit menu-item to the file-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `file-menu:after-quit` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Does nothing.

`file-menu:between-close-and-quit`

This method is called between the addition of the close menu-item and before the addition of the quit menu-item to the file-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `file-menu:between-close-and-quit` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Does nothing.

`file-menu:between-new-and-open`

This method is called between the addition of the new menu-item and before the addition of the open menu-item to the file-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `file-menu:between-new-and-open` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Does nothing.

`file-menu:between-open-and-revert`

This method is called between the addition of the open menu-item and before the addition of the revert menu-item to the file-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `file-menu:between-open-and-revert` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Does nothing.

`file-menu:between-print-and-close`

This method is called between the addition of the print menu-item and before the addition of the close menu-item to the file-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `file-menu:between-print-and-close` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Adds a separator menu item.

`file-menu:between-revert-and-save`

This method is called between the addition of the revert menu-item and before the addition of the save menu-item to the file-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `file-menu:between-revert-and-save` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Does nothing.

`file-menu:between-save-as-and-print`

This method is called between the addition of the save-as menu-item and before the addition of the print menu-item to the file-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:standard-menus* `file-menu:between-save-as-and-print` *menu*) ⇒ void
  menu : (instance (derived-from `menu%`))

  Adds a separator menu item.

`file-menu:close-callback`

This method is called when the close menu-item of the file-menu menu is selected.

- (**send** *a-frame:standard-menus* `file-menu:close-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Defaultly bound to: (**lambda** (*item control*) (**when** (*can-close?*) (*on-close*) (*show* #f)) #t)

`file-menu:close-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- (**send** *a-frame:standard-menus* `file-menu:close-help-string`) ⇒ string
  Defaultly returns "(*string-constant close-info*)"

`file-menu:close-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `file-menu:close-on-demand` *item*) ⇒ void
  item : `menu-item%`
  Defaultly is this: (**lambda** (*menu-item*) (`void`))

`file-menu:close-string`

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* `file-menu:close-string`) ⇒ string
  defaultly returns "(string-constant close-menu-item)"

`file-menu:create-close?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- (**send** *a-frame:standard-menus* `file-menu:create-close?`) ⇒ boolean
  defaultly returns (lambda (item control) (when (can-close?) (on-close) (show #f)) #t)

`file-menu:create-new?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- (**send** *a-frame:standard-menus* `file-menu:create-new?`) ⇒ boolean
  defaultly returns (lambda (item control) (handler:edit-file #f) #t)

`file-menu:create-open-recent?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `file-menu:create-open-recent?`**)** ⇒ boolean
  defaultly returns (lambda (x y) (void))

`file-menu:create-open?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `file-menu:create-open?`**)** ⇒ boolean
  defaultly returns (lambda (item control) (handler:open-file) #t)

`file-menu:create-print?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `file-menu:create-print?`**)** ⇒ boolean
  defaultly returns (lambda (item control) (void))

`file-menu:create-quit?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `file-menu:create-quit?`**)** ⇒ boolean
  defaultly returns (lambda (item control) (parameterize ((exit:frame-exiting this)) (exit:exit)))

`file-menu:create-revert?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `file-menu:create-revert?`**)** ⇒ boolean
  defaultly returns (lambda (item control) (void))

`file-menu:create-save-as?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `file-menu:create-save-as?`**)** ⇒ boolean
  defaultly returns (lambda (item control) (void))

`file-menu:create-save?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- (**send** *a-frame:standard-menus* `file-menu:create-save?`) ⇒ boolean

  defaultly returns (lambda (item control) (void))

`file-menu:get-close-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `file-menu:get-close-item`) ⇒ (instance `menu-item%`)

`file-menu:get-new-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `file-menu:get-new-item`) ⇒ (instance `menu-item%`)

`file-menu:get-open-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `file-menu:get-open-item`) ⇒ (instance `menu-item%`)

`file-menu:get-open-recent-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `file-menu:get-open-recent-item`) ⇒ (instance `menu-item%`)

`file-menu:get-print-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `file-menu:get-print-item`) ⇒ (instance `menu-item%`)

`file-menu:get-quit-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `file-menu:get-quit-item`) ⇒ (instance `menu-item%`)

`file-menu:get-revert-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `file-menu:get-revert-item`) ⇒ (instance `menu-item%`)

`file-menu:get-save-as-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `file-menu:get-save-as-item`) ⇒ (instance `menu-item%`)

`file-menu:get-save-item`

This method returns the `menu-item%` that corresponds to this menu item.

- (**send** *a-frame:standard-menus* `file-menu:get-save-item`) ⇒ (instance `menu-item%`)

`file-menu:new-callback`

This method is called when the new menu-item of the file-menu menu is selected.

- (**send** *a-frame:standard-menus* `file-menu:new-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Defaultly bound to: (**lambda** (*item control*) (*handler:edit-file* #f) #t)

`file-menu:new-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- (**send** *a-frame:standard-menus* `file-menu:new-help-string`) ⇒ string
  Defaultly returns "(*string-constant new-info*)"

`file-menu:new-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `file-menu:new-on-demand` *item*) ⇒ void
  item : `menu-item%`
  Defaultly is this: (**lambda** (*menu-item*) (`void`))

`file-menu:new-string`

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* `file-menu:new-string`) ⇒ string
  defaultly returns "(string-constant new-menu-item)"

`file-menu:open-callback`

This method is called when the open menu-item of the file-menu menu is selected.

- (**send** *a-frame:standard-menus* `file-menu:open-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Defaultly bound to: (**lambda** (*item control*) (*handler:open-file*) #t)

`file-menu:open-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- (**send** *a-frame:standard-menus* `file-menu:open-help-string`) ⇒ string
  Defaultly returns "(*string-constant open-info*)"

`file-menu:open-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `file-menu:open-on-demand` *item*) ⇒ void
  item : `menu-item%`
  Defaultly is this: (**lambda** (*menu-item*) (`void`))

`file-menu:open-recent-callback`

This method is called when the open-recent menu-item of the file-menu menu is selected.

- (**send** *a-frame:standard-menus* `file-menu:open-recent-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Defaultly bound to: (**lambda** (*x y*) (`void`))

`file-menu:open-recent-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- (**send** *a-frame:standard-menus* `file-menu:open-recent-help-string`) ⇒ string
  Defaultly returns "(*string-constant open-recent-info*)"

`file-menu:open-recent-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `file-menu:open-recent-on-demand` *item*) ⇒ void
  item : `menu-item%`
  Defaultly is this: (**lambda** (*menu*) (*handler:install-recent-items menu*))

`file-menu:open-recent-string`

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* file-menu:open-recent-string) ⇒ string

  defaultly returns "(string-constant open-recent-menu-item)"

file-menu:open-string

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* file-menu:open-string) ⇒ string

  defaultly returns "(string-constant open-menu-item)"

file-menu:print-callback

This method is called when the print menu-item of the file-menu menu is selected.

- (**send** *a-frame:standard-menus* file-menu:print-callback *item evt*) ⇒ void
  item : (instance (derived-from menu-item%))
  evt : (instance control-event%)

  Defaultly bound to: (**lambda** (*item control*) (void))

file-menu:print-help-string

This result of this method is used as the help string when the menu-item% object is created.

- (**send** *a-frame:standard-menus* file-menu:print-help-string) ⇒ string

  Defaultly returns "(*string-constant print-info*)"

file-menu:print-on-demand

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* file-menu:print-on-demand *item*) ⇒ void
  item : menu-item%

  Defaultly is this: (**lambda** (*menu-item*) (void))

file-menu:print-string

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* file-menu:print-string) ⇒ string

  defaultly returns "(string-constant print-menu-item)"

file-menu:quit-callback

This method is called when the quit menu-item of the file-menu menu is selected.

- (**send** *a-frame:standard-menus* `file-menu:quit-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Defaultly bound to: (**lambda** (*item control*) (**parameterize** ((*exit:frame-exiting* `this`)) (*exit:exit*)))

`file-menu:quit-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- (**send** *a-frame:standard-menus* `file-menu:quit-help-string`) ⇒ string
  Defaultly returns "(*string-constant quit-info*)"

`file-menu:quit-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `file-menu:quit-on-demand` *item*) ⇒ void
  item : `menu-item%`
  Defaultly is this: (**lambda** (*menu-item*) (`void`))

`file-menu:quit-string`

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* `file-menu:quit-string`) ⇒ string
  defaultly returns "(if (eq? (system-type) (quote windows)) (string-constant quit-menu-item-windows) (string-constant quit-menu-item-others))"

`file-menu:revert-callback`

This method is called when the revert menu-item of the file-menu menu is selected.

- (**send** *a-frame:standard-menus* `file-menu:revert-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Defaultly bound to: (**lambda** (*item control*) (`void`))

`file-menu:revert-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- (**send** *a-frame:standard-menus* `file-menu:revert-help-string`) ⇒ string
  Defaultly returns "(*string-constant revert-info*)"

`file-menu:revert-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* file-menu:revert-on-demand *item*) ⇒ void
  item : menu-item%

  Defaultly is this: (**lambda** (*menu-item*) (void))

file-menu:revert-string

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* file-menu:revert-string) ⇒ string

  defaultly returns "(string-constant revert-menu-item)"

file-menu:save-as-callback

This method is called when the save-as menu-item of the file-menu menu is selected.

- (**send** *a-frame:standard-menus* file-menu:save-as-callback *item evt*) ⇒ void
  item : (instance (derived-from menu-item%))
  evt : (instance control-event%)

  Defaultly bound to: (**lambda** (*item control*) (void))

file-menu:save-as-help-string

This result of this method is used as the help string when the menu-item% object is created.

- (**send** *a-frame:standard-menus* file-menu:save-as-help-string) ⇒ string

  Defaultly returns "(*string-constant save-as-info*)"

file-menu:save-as-on-demand

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* file-menu:save-as-on-demand *item*) ⇒ void
  item : menu-item%

  Defaultly is this: (**lambda** (*menu-item*) (void))

file-menu:save-as-string

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* file-menu:save-as-string) ⇒ string

  defaultly returns "(string-constant save-as-menu-item)"

file-menu:save-callback

This method is called when the save menu-item of the file-menu menu is selected.

- (**send** *a-frame:standard-menus* `file-menu:save-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Defaultly bound to: (**lambda** (*item control*) (`void`))

`file-menu:save-help-string`

This result of this method is used as the help string when the `menu-item%` object is created.

- (**send** *a-frame:standard-menus* `file-menu:save-help-string`) ⇒ string
  Defaultly returns "(*string-constant save-info*)"

`file-menu:save-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* `file-menu:save-on-demand` *item*) ⇒ void
  item : `menu-item%`
  Defaultly is this: (**lambda** (*menu-item*) (`void`))

`file-menu:save-string`

The result of this method is the name of this menu.

- (**send** *a-frame:standard-menus* `file-menu:save-string`) ⇒ string
  defaultly returns "(string-constant save-menu-item)"

`get-checkable-menu-item%`

The result of this method is used as the class for creating checkable menu items in this class (see `frame:standard-menus%` for a list).

- (**send** *a-frame:standard-menus* `get-checkable-menu-item%`) ⇒ (derived-from `checkable-menu-item%`)
  defaultly returns `menu:can-restore-checkable-menu-item%`.

`get-edit-menu`

Returns the edit menu See also `get-menu%`

- (**send** *a-frame:standard-menus* `get-edit-menu`) ⇒ (instance (derived-from `menu%`))

`get-file-menu`

Returns the file menu See also `get-menu%`

- (**send** *a-frame:standard-menus* `get-file-menu`) ⇒ (instance (derived-from `menu%`))

get-help-menu

Returns the help menu See also get-menu%

- (**send** *a-frame:standard-menus* get-help-menu) ⇒ (instance (derived-from menu%))

get-menu-item%

The result of this method is used as the class for creating the menu items in this frame (see frame:standard-menus% for a list).

- (**send** *a-frame:standard-menus* get-menu-item%) ⇒ (derived-from menu-item%)

  defaultly returns menu:can-restore-menu-item%.

get-menu%

The result of this method is used as the class for creating the result of these methods: get-file-menu, get-edit-menu, get-help-menu.

- (**send** *a-frame:standard-menus* get-menu%) ⇒ (derived-from menu:can-restore-underscore-menu%)

  defaultly returns menu%

help-menu:about-callback

This method is called when the about menu-item of the help-menu menu is selected.

- (**send** *a-frame:standard-menus* help-menu:about-callback *item evt*) ⇒ void
  item : (instance (derived-from menu-item%))
  evt : (instance control-event%)

  Defaultly bound to: (**lambda** (*item control*) (void))

help-menu:about-help-string

This result of this method is used as the help string when the menu-item% object is created.

- (**send** *a-frame:standard-menus* help-menu:about-help-string) ⇒ string

  Defaultly returns "(*string-constant about-info*)"

help-menu:about-on-demand

The menu item's on-demand method calls this method

- (**send** *a-frame:standard-menus* help-menu:about-on-demand *item*) ⇒ void
  item : menu-item%

  Defaultly is this: (**lambda** (*menu-item*) (void))

`help-menu:about-string`

The result of this method is the name of this menu.

- **(send** *a-frame:standard-menus* `help-menu:about-string`**)** ⇒ string

  defaultly returns "(string-constant about-menu-item)"

`help-menu:after-about`

This method is called after the addition of the about menu-item to the help-menu menu. Override it to add additional menus at that point.

- **(send** *a-frame:standard-menus* `help-menu:after-about` *menu***)** ⇒ void
  menu : (instance (derived-from `menu%`))

  Does nothing.

`help-menu:before-about`

This method is called before the addition of the about menu-item to the help-menu menu. Override it to add additional menus at that point.

- **(send** *a-frame:standard-menus* `help-menu:before-about` *menu***)** ⇒ void
  menu : (instance (derived-from `menu%`))

  Does nothing.

`help-menu:create-about?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:standard-menus* `help-menu:create-about?`**)** ⇒ boolean

  defaultly returns (lambda (item control) (void))

`help-menu:get-about-item`

This method returns the `menu-item%` that corresponds to this menu item.

- **(send** *a-frame:standard-menus* `help-menu:get-about-item`**)** ⇒ (instance `menu-item%`)

11.10 `frame:standard-menus-mixin`

Domain: `frame:basic<%>`

Implements: `frame:basic<%>`

Implements: frame:standard-menus<%>

This frame provides a skeleton for the standard set of menus in a frame.

The result of this mixin uses the same initialization arguments as the mixin's argument.

on-close

Called just before the window is closed (e.g., by the window manager). This method is *not* called by show .

See also can-close?.

- (**send** *a-frame:standard-menus-mixin* on-close) ⇒ void
  Removes the preferences callbacks for the menu items

11.11 frame:editor<%>

Extends: frame:standard-menus<%>

Frame classes matching this interface support embedded editors.

get-canvas

Returns the canvas used to display the editor<%> in this frame.

- (**send** *a-frame:editor* get-canvas) ⇒ (instance (derived-from canvas%))

get-canvas<%>

The result of this method is used to guard the result of the get-canvas% method.

- (**send** *a-frame:editor* get-canvas<%>) ⇒ (instance canvas:basic%)

get-canvas%

The result of this method is used to create the canvas for the editor<%> in this frame.

- (**send** *a-frame:editor* get-canvas%) ⇒ (derived-from editor-canvas%)
  Returns editor-canvas%.

get-editor

Returns the editor in this frame.

- (**send** *a-frame:editor* get-editor) ⇒ (instance (implements editor<%>))

`get-editor<%>`

The result of this method is used by `make-editor` to check that `get-editor%` is returning a reasonable editor.

- **(send** *a-frame:editor* `get-editor<%>`**)** ⇒ interface
  Returns `editor<%>`.

`get-editor%`

The result of this class is used to create the `editor<%>` in this frame.

Override this method to specify a different editor class.

- **(send** *a-frame:editor* `get-editor%`**)** ⇒ (implements `editor<%>`)

`get-entire-label`

This method returns the entire label for the frame. See also `set-label` and `set-label-prefix`.

- **(send** *a-frame:editor* `get-entire-label`**)** ⇒ string

`get-label-prefix`

This returns the prefix for the frame's label.

- **(send** *a-frame:editor* `get-label-prefix`**)** ⇒ string

`make-editor`

This method is called to create the editor in this frame. It calls `get-editor<%>` and uses that interface to make sure the result of `get-editor%` is reasonable.

- **(send** *a-frame:editor* `make-editor`**)** ⇒ (instance (implements `editor<%>`))
  Calls `(make-object get-editor%)`.

`save`

Saves the file being edited, possibly calling `save-as` if the editor has no filename yet.

- **(send** *a-frame:editor* `save` *format*`)` ⇒ boolean
  format = 'same : (union 'guess 'standard 'text 'text-force-cr 'same 'copy)
  Returns `#f` if the user cancels this operation (only possible when the file has not been saved before and the user is prompted for a new filename) and returns `#t` if not.

`save-as`

Queries the use for a file name and saves the file with that name.

- (**send** *a-frame:editor* save-as *format*) ⇒ boolean
  format = 'same : (union 'guess 'standard 'text 'text-force-cr 'same 'copy)

  Returns #f if the user cancells the file-choosing dialog and returns #t otherwise.

set-label-prefix

Sets the prefix for the label of the frame.

- (**send** *a-frame:editor* set-label-prefix *prefix*) ⇒ void
  prefix : string

11.12 frame:editor-mixin

Domain: frame:standard-menus<%>

Implements: frame:standard-menus<%>

Implements: frame:editor<%>

This mixin adds functionality to support an editor<%> in the frame. This includes management of the title, implementations of some of the menu items, an reasonable initial size, and access to the editor<%> itself.

The size of this frame with be either 600 by 650 or 65 less than the width and height of the screen, whichever is smaller.

edit-menu:between-select-all-and-find

This method is called between the addition of the select-all menu-item and before the addition of the find menu-item to the edit-menu menu. Override it to add additional menus at that point.

- (**send** *a-frame:editor-mixin* edit-menu:between-select-all-and-find *edit-menu*) ⇒ void
  edit-menu : (instance menu%)

  Adds a menu item for toggling auto-wrap in the focus'd text.

file-menu:create-print?

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- (**send** *a-frame:editor-mixin* file-menu:create-print?) ⇒ boolean

  returns #t

file-menu:create-revert?

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- (**send** *a-frame:editor-mixin* `file-menu:create-revert?`) ⇒ boolean

  returns #t

`file-menu:create-save-as?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- (**send** *a-frame:editor-mixin* `file-menu:create-save-as?`) ⇒ boolean

  returns #t

`file-menu:create-save?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- (**send** *a-frame:editor-mixin* `file-menu:create-save?`) ⇒ boolean

  returns #t

`file-menu:print-callback`

This method is called when the print menu-item of the file-menu menu is selected.

- (**send** *a-frame:editor-mixin* `file-menu:print-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)

  Calls the `print` method of `editor<%>` with the default arguments, except that the *output-mode* argument is the result of calling `preferences:get` with `'framework:print-output-mode`.

`file-menu:revert-callback`

This method is called when the revert menu-item of the file-menu menu is selected.

- (**send** *a-frame:editor-mixin* `file-menu:revert-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)

  Loads the most recently saved version of the file to the disk. If the `editor<%>` is a `text%`, the start and end positions are restored.

`file-menu:revert-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:editor-mixin* `file-menu:revert-on-demand`) ⇒ void

  Disables the menu item when the editor is locked.

`file-menu:save-as-callback`

This method is called when the save-as menu-item of the file-menu menu is selected.

- **(send** *a-frame:editor-mixin* `file-menu:save-as-callback` *item evt***)** ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)

  Prompts the user for a file name and uses that filename to save the buffer. Calls `save-as` with no arguments.

`file-menu:save-callback`

This method is called when the save menu-item of the file-menu menu is selected.

- **(send** *a-frame:editor-mixin* `file-menu:save-callback` *item evt***)** ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)

  Saves the file in the editor.

`get-filename`

This returns the filename that the frame is currently being saved as, or `#f` if there is no appropriate filename.

- **(send** *a-frame:editor-mixin* `get-filename`**)** ⇒ (union #f string)

  Returns the filename in the editor returned by `get-editor`.

`get-label`

Gets a window's label. Control windows generally display their label in some way. Frames and dialogs display their label as a window title. Panels do not display their label, but the label can be used for identification purposes. Buttons and check boxes can have bitmap labels (only when they are created with bitmap labels), but all other windows have string labels.

The label string may contain ampersands ("&"), which serve as keyboard navigation annotations for controls under Windows and X. The ampersands are not part of the displayed label of a control; instead, ampersands are removed in the displayed label (under all platforms), and any character preceding an ampersand is underlined (Windows and X) indicating that the character is a mnemonic for the control. Double ampersands are converted into a single ampersand (with no displayed underline). See also `on-traverse-char`.

If the window does not have a label, #f is returned.

- **(send** *a-frame:editor-mixin* `get-label`**)** ⇒ string

  Returns the portion of the label after the hyphen. See also `get-entire-label`.

`help-menu:about-callback`

This method is called when the about menu-item of the help-menu menu is selected.

- **(send** *a-frame:editor-mixin* `help-menu:about-callback` *item evt***)** ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)

  Calls `message-box` with a message welcoming the user to the application named by `application:current-app-name`

`help-menu:about-string`

The result of this method is the name of this menu.

- **(send** *a-frame:editor-mixin* `help-menu:about-string`**)** ⇒ string

  Returns the result of (`application:current-app-name`)

`help-menu:create-about?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:editor-mixin* `help-menu:create-about?`**)** ⇒ boolean

  returns #t

`on-close`

Called just before the window is closed (e.g., by the window manager). This method is *not* called by `show` .

See also `can-close?`.

- **(send** *a-frame:editor-mixin* `on-close`**)** ⇒ void

  Calls the `editor:basic<%>`'s method `on-close`.

`set-label`

Sets a window's label. The window's natural minimum size might be different after the label is changed, but the window's minimum size is not recomputed.

See `get-label` for more information.

- **(send** *a-frame:editor-mixin* `set-label` *l***)** ⇒ void
  l : string (up to 200 characters) or #f

  If *l* is #f, the window's label is removed.

  Sets the label, but preserve's the label's prefix. See also `set-label-prefix`.

11.13 `frame:open-here<%>`

Extends: `frame:editor<%>`

Frames implementing this mixin can change the file they are displaying.

The frame is only re-used when the 'framework:open-here? preference is set (see preferences:get and preferences:set for details on preferences).

The frame:open-here-mixin implements this interface.

get-open-here-editor

When the user switches the visible file in this frame, the of this method is the editor that gets switched.

- (**send** *a-frame:open-here* get-open-here-editor) ⇒ (is-a?/c editor¡%¿)
  Defaultly returns the result of get-editor.

open-here

- (**send** *a-frame:open-here* open-here *filename*) ⇒ void
  filename : string
  Opens *filename* in the current frame, possibly prompting the user about saving a file (in which case the frame might not get switched).

11.14 frame:open-here-mixin

Domain: frame:editor<%>

Implements: frame:open-here<%>

Implements: frame:editor<%>

Provides an implementation of frame:open-here<%>

file-menu:new-callback

This method is called when the new menu-item of the file-menu menu is selected.

- (**send** *a-frame:open-here-mixin* file-menu:new-callback *item* *evt*) ⇒ void
  item : (instance (derived-from menu-item%))
  evt : (instance control-event%)
  When the preference 'framework:open-here? preference is set, this method prompts the user, asking if they would like to create a new frame, or just clear out this one. If they clear it out and the file hasn't been saved, they are asked about saving.

file-menu:new-on-demand

The menu item's on-demand method calls this method

- (**send** *a-frame:open-here-mixin* file-menu:new-on-demand *item*) ⇒ void
  item : (is-a?/c menu-item%)

Defaultly is this: (**lambda** (*menu-item*) (`void`))

Sets the label of *item* to "New..." if the preference 'framework:open-here? is set.

`file-menu:open-on-demand`

The menu item's on-demand method calls this method

- (**send** *a-frame:open-here-mixin* `file-menu:open-on-demand` *item*) ⇒ void
  item : (is-a?/c `menu-item%`)

  Defaultly is this: (**lambda** (*menu-item*) (`void`))

  Sets the label of *item* to "Open Here..." if the preference 'framework:open-here? is set.

`on-activate`

Called when a window is *activated* or *deactivated*. A top-level window is activated when the keyboard focus moves from outside the window to the window or one of its children. It is deactivated when the focus moves back out of the window.

The method's argument is #t when the window is activated, #f when it is deactivated.

- (**send** *a-frame:open-here-mixin* `on-activate` *on?*) ⇒ void
  on? : boolean

  When *on?* is #t, calls `set-open-here-frame` with `this`.

`on-close`

Called just before the window is closed (e.g., by the window manager). This method is *not* called by `show` .

See also `can-close?`.

- (**send** *a-frame:open-here-mixin* `on-close`) ⇒ void

  Calls `set-open-here-frame` with #f if the result of `get-open-here-frame` is `eq?` to `this`.

11.15 `frame:text<%>`

Extends: `frame:editor<%>`

Frames matching this interface provide support for `text%`s.

11.16 `frame:text-mixin`

Domain: `frame:editor<%>`

Implements: `frame:editor<%>`

Implements: `frame:text<%>`

This mixins adds support for `text%`s in the frame.

The result of this mixin uses the same initialization arguments as the mixin's argument.

`get-editor<%>`

The result of this method is used by `make-editor` to check that `get-editor%` is returning a reasonable editor.

- (**send** *a-frame:text-mixin* `get-editor<%>`) ⇒ interface
  Returns `(class->interface text%)`.

`get-editor%`

The result of this class is used to create the `editor<%>` in this frame.

Override this method to specify a different editor class.

- (**send** *a-frame:text-mixin* `get-editor%`) ⇒ (implements `editor<%>`)
  Returns `text:keymap%`.

11.17 `frame:pasteboard<%>`

Extends: `frame:editor<%>`

Frames matching this interface provide support for `pasteboard%`s.

11.18 `frame:pasteboard-mixin`

Domain: `frame:editor<%>`

Implements: `frame:pasteboard<%>`

Implements: `frame:editor<%>`

This mixin provides support for pasteboards in a frame.

The result of this mixin uses the same initialization arguments as the mixin's argument.

`get-editor<%>`

The result of this method is used by `make-editor` to check that `get-editor%` is returning a reasonable editor.

- (**send** *a-frame:pasteboard-mixin* `get-editor<%>`) ⇒ interface
  Returns `(class->interface pasteboard%)`.

`get-editor%`

The result of this class is used to create the `editor<%>` in this frame.

Override this method to specify a different editor class.

- (**send** *a-frame:pasteboard-mixin* `get-editor%`) ⇒ (implements `editor<%>`)
  Returns `pasteboard:keymap%`.

11.19 `frame:delegate<%>`

Extends: `frame:text<%>`

Frames that implement this interface provide a 20,000 feet overview of the text in the main editor. The term **delegate** in these method descriptions refers to the original editor and the term **delegatee** refers to the editor showing the 20,000 feet overview.

`delegate-moved`

This method is called when the visible region of the delegate editor changes, so that the blue region in the delegatee is updated.

- (**send** *a-frame:delegate* `delegate-moved`) ⇒ void

`delegated-text-shown?`

Returns `#t` if the delegate is visible, and `#f` if it isn't.

- (**send** *a-frame:delegate* `delegated-text-shown?`) ⇒ boolean

`get-delegated-text`

Returns the delegate text.

- (**send** *a-frame:delegate* `get-delegated-text`) ⇒ (instanceof (implements `text:delegate<%>`))

`hide-delegated-text`

Hides the delegated text.

When the delegated text is hidden, it is not being updated. This is accomplished by calling the `set-delegate` method of `get-editor`with `#f`.

See also `show-delegated-text`

- (**send** *a-frame:delegate* hide-delegated-text) ⇒ void

hide-delegated-text

Makes the delegated text visible.

When the delegated text is shown, the set-delegate method of get-delegated-textis called with the text to delegate messages to.

See also hide-delegated-text.

- (**send** *a-frame:delegate* show-delegated-text) ⇒ void

11.20 frame:delegate-mixin

Domain: frame:text<%>

Implements: frame:delegate<%>

Implements: frame:text<%>

Adds support for a 20,000-feet view via text:delegate<%> and text:delegate-mixin

get-editor<%>

The result of this method is used by make-editor to check that get-editor% is returning a reasonable editor.

- (**send** *a-frame:delegate-mixin* get-editor<%>) ⇒ interface
  Returns editor<%>.
  Returns text:delegate<%>.

get-editor%

The result of this class is used to create the editor<%> in this frame.

Override this method to specify a different editor class.

- (**send** *a-frame:delegate-mixin* get-editor%) ⇒ (implements text:delegate<%>)
  returns the super rsult, with the text:delegate-mixin mixed in.

make-root-area-container

Override this method to insert a panel in between the panel used by the clients of this frame and the frame itself. For example, to insert a status line panel override this method with something like this:

(**class** ...

```
  ...
(rename [super-make-root-area-container make-root-area-container])
(field
  [status-panel #f])
(define/override (make-root-area-container cls parent)
  (set! status-panel
        (super-make-root-area-container vertical-panel% parent))
  (let ([root (make-object cls status-panel)])

    ; ... add other children to status-panel ...

    root))
  ...
```

In this example, status-panel will contain a root panel for the other classes, and whatever panels are needed to display status information.

The searching frame is implemented using this method.

- (**send** *a-frame:delegate-mixin* `make-root-area-container` *class parent*) ⇒ (implements `panel%`)
  class : (derived-from `panel%`)
  parent : (instanceof (derived-from `panel%`))

  Calls `make-object` with *class* and *parent*.

  adds a panel outside to hold the delegate `editor-canvas%` and `text%`.

11.21 `frame:searchable<%>`

Extends: `frame:basic<%>`

Frames that implement this interface support searching.

`can-replace?`

Returns `#t` if a replace command would succeed.

- (**send** *a-frame:searchable* `can-replace?`) ⇒ boolean

  Defaultly is `#t` when the selected text in the result of `get-text-to-search` is the same as the text in the find text.

`get-text-to-search`

Override this method to specify which text to search.

- (**send** *a-frame:searchable* `get-text-to-search`) ⇒ (instance (derived-from `text%`))

  Returns the result of `get-editor`.

`hide-search`

This method hides the searching information on the bottom of the frame.

- **(send** *a-frame:searchable* `hide-search`**)** ⇒ void

`move-to-search-or-reverse-search`

This method moves the focus to the text that contains the search string, or if the focus is there already, performs a reverse search.

It returns void if the focus was not to the search text, otherwise it returns a boolean indicating the success of the search.

- **(send** *a-frame:searchable* `move-to-search-or-reverse-search`**)** ⇒ (union boolean void)

`move-to-search-or-search`

This method moves the focus to the text that contains the search string, or if the focus is there already, performs a forward search.

It returns void if the focus was not to the search text, otherwise it returns a boolean indicating the success of the search.

- **(send** *a-frame:searchable* `move-to-search-or-search`**)** ⇒ (union boolean void)

`replace`

If the selected text matches the search string, this method replaces the text with the contents of the replace text. If the replace was successful, `#t` is returned. Otherwise, `#f` is returned.

- **(send** *a-frame:searchable* `replace`**)** ⇒ boolean

`replace-all`

Loops through the text from the current position to the end, replacing all occurrances of the search string with the contents of the replace edit. Only searches forward, does not loop around to the beginning of the text.

- **(send** *a-frame:searchable* `replace-all`**)** ⇒ void

`replace&search`

Calls `replace` and if it returns `#t`, calls `search-again`.

- **(send** *a-frame:searchable* `replace&search`**)** ⇒ boolean

`search-again`

Searches for the text in the search edit in the result of `get-text-to-search`.

- **(send** *a-frame:searchable* `search-again` *direction beep?*) ⇒ boolean
  direction = previous searching direction : 'forward or 'backward
  beep? = #t : bool

  Returns `#t` if the text is found and sets the selection to the found text. If the text is not found it returns `#f`.

`set-search-direction`

Sets the direction that future searches will be performed.

- **(send** *a-frame:searchable* `set-search-direction` *dir*) ⇒ void
  dir : (union -1 1)

  If *dir* is `1` searches will be performed forwards and if *dir* is `-1` searches will be performed backwards.

`toggle-search-focus`

Toggles the keyboard focus between the searching edit, the replacing edit and the result of `get-text-to-search`.

- **(send** *a-frame:searchable* `toggle-search-focus`) ⇒ void

`unhide-search`

When the searching sub window is hidden, makes it visible.

- **(send** *a-frame:searchable* `unhide-search`) ⇒ void

11.22 `frame:searchable-mixin`

Domain: `frame:standard-menus<%>`

Implements: `frame:standard-menus<%>`

Implements: `frame:searchable<%>`

This mixin adds support for searching in the `editor<%>` in this frame.

The result of this mixin uses the same initialization arguments as the mixin's argument.

`edit-menu:create-find-again?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:searchable-mixin* `edit-menu:create-find-again?`) ⇒ boolean

  returns #t

`edit-menu:create-find?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:searchable-mixin* `edit-menu:create-find?`**)** ⇒ boolean
  returns #t

`edit-menu:create-replace-and-find-again?`

The result of this method determines if the corresponding menu-item is created. Override this to control the creation of the menu-item.

- **(send** *a-frame:searchable-mixin* `edit-menu:create-replace-and-find-again?`**)** ⇒ boolean
  returns #t

`edit-menu:find-again-callback`

This method is called when the find-again menu-item of the edit-menu menu is selected.

- **(send** *a-frame:searchable-mixin* `edit-menu:find-again-callback`**)** ⇒ boolean
  Returns `#t`, and searches for the same text that was last searched for in the text.

`edit-menu:find-callback`

This method is called when the find menu-item of the edit-menu menu is selected.

- **(send** *a-frame:searchable-mixin* `edit-menu:find-callback` *item evt*) ⇒ void
  item : (instance (derived-from `menu-item%`))
  evt : (instance `control-event%`)
  Calls `move-to-search-or-search`.

`edit-menu:replace-and-find-again-callback`

This method is called when the replace-and-find-again menu-item of the edit-menu menu is selected.

- **(send** *a-frame:searchable-mixin* `edit-menu:replace-and-find-again-callback`**)** ⇒ boolean
  Returns `#t`, and if the selected text matches the current text in the find box, replaces it with the contents of the replace box and searches for the next occurance of the text in the find box.

`edit-menu:replace-and-find-again-on-demand`

The menu item's on-demand method calls this method

- **(send** *a-frame:searchable-mixin* `edit-menu:replace-and-find-again-on-demand` *item*) ⇒ void
  item : `menu-item%`
  Disables *item* when `can-replace?` returns `#f` and enables it when that method returns `#t`.

`make-root-area-container`

Override this method to insert a panel in between the panel used by the clients of this frame and the frame itself. For example, to insert a status line panel override this method with something like this:

```
(class ...
  ...
 (rename [super-make-root-area-container make-root-area-container])
 (field
   [status-panel #f])
 (define/override (make-root-area-container cls parent)
   (set! status-panel
         (super-make-root-area-container vertical-panel% parent))
   (let ([root (make-object cls status-panel)])

     ; ... add other children to status-panel ...

     root))
  ...
```

In this example, status-panel will contain a root panel for the other classes, and whatever panels are needed to display status information.

The searching frame is implemented using this method.

- **(send** *a-frame:searchable-mixin* `make-root-area-container`**)** ⇒ (implements `area-container<%>`)

  Calls `make-object` with *class* and *parent*.

  Builds a panel for the searching information.

`on-activate`

Called when a window is *activated* or *deactivated*. A top-level window is activated when the keyboard focus moves from outside the window to the window or one of its children. It is deactivated when the focus moves back out of the window.

The method's argument is #t when the window is activated, #f when it is deactivated.

- **(send** *a-frame:searchable-mixin* `on-activate` *active?***)** ⇒ void
  active? : boolean

  When the frame is activated, searches will take place in this frame.

`on-close`

Called just before the window is closed (e.g., by the window manager). This method is *not* called by `show` .

See also `can-close?`.

- **(send** *a-frame:searchable-mixin* `on-close`**)** ⇒ void

  Cleans up after the searching frame.

11.23 frame:searchable-text<%>

Extends: frame:text<%>

Extends: frame:searchable<%>

11.24 frame:searchable-text-mixin

Domain: frame:text<%>

Domain: frame:searchable<%>

Implements: frame:searchable-text<%>

Implements: frame:text<%>

Implements: frame:searchable<%>

get-editor<%>

The result of this method is used by make-editor to check that get-editor% is returning a reasonable editor.

- (**send** *a-frame:searchable-text-mixin* get-editor<%>) ⇒ (implements editor<%>)

  Returns text:searching<%>.

get-editor%

The result of this class is used to create the editor<%> in this frame.

Override this method to specify a different editor class.

- (**send** *a-frame:searchable-text-mixin* get-editor%) ⇒ (implements editor<%>)

  Returns text:searching%.

get-text-to-search

Override this method to specify which text to search.

- (**send** *a-frame:searchable-text-mixin* get-text-to-search) ⇒ (instanceof text%)

  Returns the result of get-editor.

11.25 `frame:file<%>`

Extends: `frame:editor<%>`

Frames supporting this interface manage editors, like frames supporting the `frame:editor<%>` interface, but in addition, they support editors that match the `editor:file<%>` interface.

11.26 `frame:file-mixin`

Domain: `frame:editor<%>`

Implements: `frame:editor<%>`

Implements: `frame:file<%>`

This mixin adds support for `editor:file<%>` objects.

The result of this mixin uses the same initialization arguments as the mixin's argument.

`can-close?`

Called just before the window might be closed (e.g., by the window manager). If #f is returned, the window is not closed, otherwise `on-close` is called and the window is closed (i.e., the window is hidden, like calling `show` with #f).

This method is *not* called by `show` .

- **(send** *a-frame:file-mixin* `can-close?`**)** ⇒ boolean

  Checks to see if the editor has been saved.

11.27 `frame:basic% = (frame:basic-mixin frame%)`

```
frame:basic% = (frame:basic-mixin frame%)
```

- **(instantiate** `frame:basic%` **()** **(label** _**)** **[(parent** _**)]** **[(width** _**)]** **[(height** _**)]** **[(x** _**)]** **[(y** _**)]** **[(style** _**)]** **[(enabled** _**)]** **[(border** _**)]** **[(spacing** _**)]** **[(alignment** _**)]** **[(min-width** _**)]** **[(min-height** _**)]** **[(stretchable-width** _**)]** **[(stretchable-height** _**)])** ⇒ `frame:basic%` object
  label : string (up to 200 characters)
  parent = #f : `frame%` object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in `'(no-resize-border no-caption no-system-menu mdi-parent mdi-child)`
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]

spacing = 0 : exact integer in [0, 1000]
alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
min-width = 0 : exact integer in [0, 10000]
min-height = 0 : exact integer in [0, 10000]
stretchable-width = #t : boolean
stretchable-height = #t : boolean

The *label* string is displayed in the frame's title bar. If the frame's label is changed (see set-label), the title bar is updated.

The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by current-eventspace . Otherwise, *parent*'s eventspace is the new frame's eventspace.

If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

The *style* flags adjust the appearance of the frame on some platforms:

- 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
- 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
- 'no-system-menu — omits the system menu (Windows)
- 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
- 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an exn:application:mismatch exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see window<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

11.28 frame:info% = (frame:info-mixin frame:basic%)

frame:info% = (frame:info-mixin frame:basic%)

- **(instantiate** frame:info% () **(label** _**)** [**(parent** _**)**] [**(width** _**)**] [**(height** _**)**] [**(x** _**)**] [**(y** _**)**] [**(style** _**)**] [**(enabled** _**)**] [**(border** _**)**] [**(spacing** _**)**] [**(alignment** _**)**] [**(min-width** _**)**] [**(min-height** _**)**] [**(stretchable-width** _**)**] [**(stretchable-height** _**)**]) ⇒ frame:info% object
  label : string (up to 200 characters)
  parent = #f : frame% object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f

x = #f : exact integer in [0, 10000] or #f
y = #f : exact integer in [0, 10000] or #f
style = null : list of symbols in '(no-resize-border no-caption no-system-menu mdi-parent mdi-child)
enabled = #t : boolean
border = 0 : exact integer in [0, 1000]
spacing = 0 : exact integer in [0, 1000]
alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
min-width = 0 : exact integer in [0, 10000]
min-height = 0 : exact integer in [0, 10000]
stretchable-width = #t : boolean
stretchable-height = #t : boolean

The *label* string is displayed in the frame's title bar. If the frame's label is changed (see set-label), the title bar is updated.

The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by current-eventspace . Otherwise, *parent*'s eventspace is the new frame's eventspace.

If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

The *style* flags adjust the appearance of the frame on some platforms:

- 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
- 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
- 'no-system-menu — omits the system menu (Windows)
- 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
- 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an exn:application:mismatch exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see window<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

11.29 frame:text-info% = (frame:text-info-mixin frame:info%)

frame:text-info% = (frame:text-info-mixin frame:info%)

- **(instantiate** frame:text-info% () **(label** _**)** [**(parent** _**)**] [**(width** _**)**] [**(height** _**)**] [**(x** _**)**] [**(y** _**)**] [**(style** _**)**] [**(enabled** _**)**] [**(border** _**)**] [**(spacing** _**)**] [**(alignment** _**)**] [**(min-width**

_)] [(**min-height** _)] [(**stretchable-width** _)] [(**stretchable-height** _)]) ⇒ frame:text-info% object

label : string (up to 200 characters)
parent = #f : frame% object or #f
width = #f : exact integer in [0, 10000] or #f
height = #f : exact integer in [0, 10000] or #f
x = #f : exact integer in [0, 10000] or #f
y = #f : exact integer in [0, 10000] or #f
style = null : list of symbols in '(no-resize-border no-caption no-system-menu mdi-parent mdi-child)
enabled = #t : boolean
border = 0 : exact integer in [0, 1000]
spacing = 0 : exact integer in [0, 1000]
alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
min-width = 0 : exact integer in [0, 10000]
min-height = 0 : exact integer in [0, 10000]
stretchable-width = #t : boolean
stretchable-height = #t : boolean

The *label* string is displayed in the frame's title bar. If the frame's label is changed (see set-label), the title bar is updated.

The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by current-eventspace . Otherwise, *parent*'s eventspace is the new frame's eventspace.

If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

The *style* flags adjust the appearance of the frame on some platforms:

- 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
- 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
- 'no-system-menu — omits the system menu (Windows)
- 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
- 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an exn:application:mismatch exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see window<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

11.30 frame:pasteboard-info% = (frame:pasteboard-info-mixin frame:text-info%)

frame:pasteboard-info% = (frame:pasteboard-info-mixin frame:text-info%)

- (**instantiate** frame:pasteboard-info% () (**label** _) [(**parent** _)] [(**width** _)] [(**height** _)] [(**x** _)] [(**y** _)] [(**style** _)] [(**enabled** _)] [(**border** _)] [(**spacing** _)] [(**alignment** _)] [(**min-width** _)] [(**min-height** _)] [(**stretchable-width** _)] [(**stretchable-height** _)]) ⇒ frame:pasteboard-info object
  label : string (up to 200 characters)
  parent = #f : frame% object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in '(no-resize-border no-caption no-system-menu mdi-parent mdi-child)
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  The *label* string is displayed in the frame's title bar. If the frame's label is changed (see set-label), the title bar is updated.

  The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

  If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by current-eventspace . Otherwise, *parent*'s eventspace is the new frame's eventspace.

  If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

  If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

  The *style* flags adjust the appearance of the frame on some platforms:

  – 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
  – 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
  – 'no-system-menu — omits the system menu (Windows)
  – 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
  – 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

  If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an exn:application:mismatch exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see window<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

11.31 frame:standard-menus% = (frame:standard-menus-mixin frame:pasteboard-info%)

frame:standard-menus% = (frame:standard-menus-mixin frame:pasteboard-info%)

- **(instantiate frame:standard-menus% () (label _) [(parent _)] [(width _)] [(height _)] [(x _)] [(y _)] [(style _)] [(enabled _)] [(border _)] [(spacing _)] [(alignment _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)])** ⇒ frame:standard-menus% object
  label : string (up to 200 characters)
  parent = #f : frame% object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in '(no-resize-border no-caption no-system-menu mdi-parent mdi-child)
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  The *label* string is displayed in the frame's title bar. If the frame's label is changed (see set-label), the title bar is updated.

  The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

  If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by current-eventspace . Otherwise, *parent*'s eventspace is the new frame's eventspace.

  If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

  If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

  The *style* flags adjust the appearance of the frame on some platforms:

  – 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
  – 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)

– 'no-system-menu — omits the system menu (Windows)
– 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
– 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an `exn:application:mismatch` exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see `window<%>`. For information about the *border*, *spacing*, and *alignment* arguments, see `area-container<%>`. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see `area<%>`.

11.32 frame:editor% = (frame:editor-mixin frame:standard-menus%)

`frame:editor% = (frame:editor-mixin frame:standard-menus%)`

- **(instantiate** `frame:editor%` **() (label _) [(parent _)] [(width _)] [(height _)] [(x _)] [(y _)] [(style _)] [(enabled _)] [(border _)] [(spacing _)] [(alignment _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)])** ⇒ `frame:editor%` object

label : string (up to 200 characters)
parent = #f : `frame%` object or #f
width = #f : exact integer in [0, 10000] or #f
height = #f : exact integer in [0, 10000] or #f
x = #f : exact integer in [0, 10000] or #f
y = #f : exact integer in [0, 10000] or #f
style = null : list of symbols in `'(no-resize-border no-caption no-system-menu mdi-parent mdi-child)`
enabled = #t : boolean
border = 0 : exact integer in [0, 1000]
spacing = 0 : exact integer in [0, 1000]
alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
min-width = 0 : exact integer in [0, 10000]
min-height = 0 : exact integer in [0, 10000]
stretchable-width = #t : boolean
stretchable-height = #t : boolean

The *label* string is displayed in the frame's title bar. If the frame's label is changed (see `set-label`), the title bar is updated.

The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by `current-eventspace` . Otherwise, *parent*'s eventspace is the new frame's eventspace.

If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

The *style* flags adjust the appearance of the frame on some platforms:

- 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
- 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
- 'no-system-menu — omits the system menu (Windows)
- 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
- 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an `exn:application:mismatch` exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see `window<%>`. For information about the *border*, *spacing*, and *alignment* arguments, see `area-container<%>`. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see `area<%>`.

11.33 frame:open-here% = (frame:open-here-mixin frame:editor%)

`frame:open-here% = (frame:open-here-mixin frame:editor%)`

- **(instantiate** `frame:open-here%` **() (label _) [(parent _)] [(width _)] [(height _)] [(x _)] [(y _)] [(style _)] [(enabled _)] [(border _)] [(spacing _)] [(alignment _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)])** ⇒ `frame:open-here%`
  object
  label : string (up to 200 characters)
  parent = #f : `frame%` object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in `'(no-resize-border no-caption no-system-menu mdi-parent mdi-child)`
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  The *label* string is displayed in the frame's title bar. If the frame's label is changed (see `set-label`), the title bar is updated.

  The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by current-eventspace . Otherwise, *parent*'s eventspace is the new frame's eventspace.

If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

The *style* flags adjust the appearance of the frame on some platforms:

- 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
- 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
- 'no-system-menu — omits the system menu (Windows)
- 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
- 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an exn:application:mismatch exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see window<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

11.34 frame:text% = (frame:text-mixin frame:open-here%)

frame:text% = (frame:text-mixin frame:open-here%)

- **(instantiate** frame:text% () **(label** _**)** [**(parent** _**)**] [**(width** _**)**] [**(height** _**)**] [**(x** _**)**] [**(y** _**)**] [**(style** _**)**] [**(enabled** _**)**] [**(border** _**)**] [**(spacing** _**)**] [**(alignment** _**)**] [**(min-width** _**)**] [**(min-height** _**)**] [**(stretchable-width** _**)**] [**(stretchable-height** _**)**]**)** ⇒ frame:text% object
  label : string (up to 200 characters)
  parent = #f : frame% object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in '(no-resize-border no-caption no-system-menu mdi-parent mdi-child)
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

The *label* string is displayed in the frame's title bar. If the frame's label is changed (see set-label), the title bar is updated.

The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by current-eventspace . Otherwise, *parent*'s eventspace is the new frame's eventspace.

If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

The *style* flags adjust the appearance of the frame on some platforms:

- 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
- 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
- 'no-system-menu — omits the system menu (Windows)
- 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
- 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an exn:application:mismatch exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see window<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

11.35 frame:text-info-file% = (frame:file-mixin frame:text%)

frame:text-info-file% = (frame:file-mixin frame:text%)

- **(instantiate frame:text-info-file% () (label _) [(parent _)] [(width _)] [(height _)] [(x _)] [(y _)] [(style _)] [(enabled _)] [(border _)] [(spacing _)] [(alignment _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)]) ⇒ frame:text-info-file% object**
  label : string (up to 200 characters)
  parent = #f : frame% object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in '(no-resize-border no-caption no-system-menu mdi-parent mdi-child)
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]

spacing = 0 : exact integer in [0, 1000]
alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
min-width = 0 : exact integer in [0, 10000]
min-height = 0 : exact integer in [0, 10000]
stretchable-width = #t : boolean
stretchable-height = #t : boolean

The *label* string is displayed in the frame's title bar. If the frame's label is changed (see `set-label`), the title bar is updated.

The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by `current-eventspace` . Otherwise, *parent*'s eventspace is the new frame's eventspace.

If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

The *style* flags adjust the appearance of the frame on some platforms:

- 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
- 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
- 'no-system-menu — omits the system menu (Windows)
- 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
- 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an `exn:application:mismatch` exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see `window<%>`. For information about the *border*, *spacing*, and *alignment* arguments, see `area-container<%>`. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see `area<%>`.

11.36 `frame:searchable% = (frame:searchable-text-mixin (frame:searchable-mixin frame:text-info-file%))`

`frame:searchable% = (frame:searchable-text-mixin (frame:searchable-mixin frame:text-info-file%))`

- **(instantiate** `frame:searchable%` **() (label _) [(parent _)] [(width _)] [(height _)] [(x _)] [(y _)] [(style _)] [(enabled _)] [(border _)] [(spacing _)] [(alignment _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)])** ⇒ `frame:searchable%` object
  label : string (up to 200 characters)
  parent = #f : `frame%` object or #f

width = #f : exact integer in [0, 10000] or #f
height = #f : exact integer in [0, 10000] or #f
x = #f : exact integer in [0, 10000] or #f
y = #f : exact integer in [0, 10000] or #f
style = null : list of symbols in '(no-resize-border no-caption no-system-menu mdi-parent mdi-child)
enabled = #t : boolean
border = 0 : exact integer in [0, 1000]
spacing = 0 : exact integer in [0, 1000]
alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
min-width = 0 : exact integer in [0, 10000]
min-height = 0 : exact integer in [0, 10000]
stretchable-width = #t : boolean
stretchable-height = #t : boolean

The *label* string is displayed in the frame's title bar. If the frame's label is changed (see set-label), the title bar is updated.

The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by current-eventspace . Otherwise, *parent*'s eventspace is the new frame's eventspace.

If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

The *style* flags adjust the appearance of the frame on some platforms:

- 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
- 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
- 'no-system-menu — omits the system menu (Windows)
- 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
- 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an exn:application:mismatch exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see window<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

11.37 frame:delegate% = (frame:delegate-mixin frame:searchable%)

frame:delegate% = (frame:delegate-mixin frame:searchable%)

- **(instantiate** `frame:delegate%` **()** **(label** _**)** **[(parent** _**)]** **[(width** _**)]** **[(height** _**)]** **[(x** _**)]** **[(y** _**)]** **[(style** _**)]** **[(enabled** _**)]** **[(border** _**)]** **[(spacing** _**)]** **[(alignment** _**)]** **[(min-width** _**)]** **[(min-height** _**)]** **[(stretchable-width** _**)]** **[(stretchable-height** _**)])** ⇒ `frame:delegate%`
  object
  label : string (up to 200 characters)
  parent = #f : `frame%` object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in `'(no-resize-border no-caption no-system-menu mdi-parent mdi-child)`
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  The *label* string is displayed in the frame's title bar. If the frame's label is changed (see `set-label`), the title bar is updated.

  The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

  If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by `current-eventspace` . Otherwise, *parent*'s eventspace is the new frame's eventspace.

  If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

  If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

  The *style* flags adjust the appearance of the frame on some platforms:

  - 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
  - 'no-caption — omits the title bar for the frame (Windows, X MWM)
    (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
  - 'no-system-menu — omits the system menu (Windows)
  - 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
  - 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

  If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an `exn:application:mismatch` exception is raised.

  Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

  For information about the *enabled* argument, see `window<%>`. For information about the *border*, *spacing*, and *alignment* arguments, see `area-container<%>`. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see `area<%>`.

11.38 frame:pasteboard% = (frame:pasteboard-mixin frame:open-here%)

`frame:pasteboard% = (frame:pasteboard-mixin frame:open-here%)`

- **(instantiate frame:pasteboard% () (label _) [(parent _)] [(width _)] [(height _)] [(x _)] [(y _)] [(style _)] [(enabled _)] [(border _)] [(spacing _)] [(alignment _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)])** ⇒ frame:pasteboard% object
  label : string (up to 200 characters)
  parent = #f : frame% object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in '(no-resize-border no-caption no-system-menu mdi-parent mdi-child)
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  The *label* string is displayed in the frame's title bar. If the frame's label is changed (see set-label), the title bar is updated.

  The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

  If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by current-eventspace . Otherwise, *parent*'s eventspace is the new frame's eventspace.

  If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

  If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

  The *style* flags adjust the appearance of the frame on some platforms:

  - 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
  - 'no-caption — omits the title bar for the frame (Windows, X MWM)
    (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
  - 'no-system-menu — omits the system menu (Windows)
  - 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
  - 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

  If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an exn:application:mismatch exception is raised.

  Even if the frame is not shown, a few notification events may be queued for the frame on creation.

Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see window<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

11.39 frame:pasteboard-info-file% = (frame:file-mixin frame:pasteboard%)

frame:pasteboard-info-file% = (frame:file-mixin frame:pasteboard%)

- **(instantiate** frame:pasteboard-info-file% () **(label** _**)** [**(parent** _**)**] [**(width** _**)**] [**(height** _**)**] [**(x** _**)**] [**(y** _**)**] [**(style** _**)**] [**(enabled** _**)**] [**(border** _**)**] [**(spacing** _**)**] [**(alignment** _**)**] [**(min-width** _**)**] [**(min-height** _**)**] [**(stretchable-width** _**)**] [**(stretchable-height** _**)**]) ⇒ frame:pasteboard-info-file% object
  label : string (up to 200 characters)
  parent = #f : frame% object or #f
  width = #f : exact integer in [0, 10000] or #f
  height = #f : exact integer in [0, 10000] or #f
  x = #f : exact integer in [0, 10000] or #f
  y = #f : exact integer in [0, 10000] or #f
  style = null : list of symbols in '(no-resize-border no-caption no-system-menu mdi-parent mdi-child)
  enabled = #t : boolean
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  The *label* string is displayed in the frame's title bar. If the frame's label is changed (see set-label), the title bar is updated.

  The *parent* argument can be #f or an existing frame. Under Windows, if *parent* is an existing frame, the new frame is always on top of its parent. Also, the *parent* frame may be an MDI parent frame from a new MDI child frame. Under Windows and X (for many window managers), a frame is iconized when its parent is iconized.

  If *parent* is #f, then the eventspace for the new frame is the current eventspace, as determined by current-eventspace . Otherwise, *parent*'s eventspace is the new frame's eventspace.

  If the *width* or *height* argument is not #f, it specifies an initial size for the frame (in pixels) assuming that it is larger than the minimum size, otherwise the minimum size is used.

  If the *x* or *y* argument is not #f, it specifies an initial location for the frame. Otherwise, a location is selected automatically (tiling frames and dialogs as they are created).

  The *style* flags adjust the appearance of the frame on some platforms:

  – 'no-resize-border — omits the resizeable border around the window (Windows, X MWM) or grow box in the bottom right corner (Mac OS)
  – 'no-caption — omits the title bar for the frame (Windows, X MWM)
  (X Gnome, X KDE: the frame decoration is omitted completely when 'no-resize-border and 'no-caption are combined.)
  – 'no-system-menu — omits the system menu (Windows)

– 'mdi-child — creates the frame as a MDI (multiple document interface) child frame, mutually exclusive with 'mdi-parent (Windows)
– 'mdi-parent — creates the frame as a MDI (multiple document interface) parent frame, mutually exclusive with 'mdi-child (Windows)

If the 'mdi-child style is specified, the *parent* must be a frame with the 'mdi-parent style, otherwise an `exn:application:mismatch` exception is raised.

Even if the frame is not shown, a few notification events may be queued for the frame on creation. Consequently, the new frame's resources (e.g., memory) cannot be reclaimed until some events are handled, or the frame's eventspace is shut down.

For information about the *enabled* argument, see `window<%>`. For information about the *border*, *spacing*, and *alignment* arguments, see `area-container<%>`. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see `area<%>`.

12. Group

A *frame group* associates a group of frames together. There is one frame group in mred, called `group:the-frame-group`, which is an object of the `group:%` class.

The frame group manages the windows menu. It also and enables the close menu item on each frame, when there is more than one frame in the group, and disables the close menu item when there is only one frame in the frame group.

- group:

12.1 group:%

This class manages a group of frames matching the `frame:basic<%>` interface. There is one instance created by the framework, returned by the function `group:get-the-frame-group` and every frame that was constructed with `frame:basic-mixin` adds itself to the result of `group:get-the-frame-group`.

`can-close-all?`

Call this method to make sure that closing all of the frames in the frame groups is permitted by the user. The function `on-close-all` is expected to be called just after this method is called.

- **(send** *a-group:* `can-close-all?)` ⇒ void
  Calls the `can-close?` method of each frame in the group.

`can-remove-frame?`

- **(send** *a-group:* `can-remove-frame?)` ⇒ boolean

`clear`

This removes all of the frames in the group. It does not close the frames. See also `on-close-all`and `can-close-all?`.

- **(send** *a-group:* `clear)` ⇒ boolean

`for-each-frame`

This method applies a function to each frame in the group. It also remembers the function and applies it to any new frames that are added to the group when they are added.

See also `get-frames`.

- (**send** *a-group:* for-each-frame *f*) ⇒ void
  f : ((instance frame:basic<%>) -¿ void)

  Applies *f* to each frame in the group

frame-label-changed

This method is called by frames constructed with frame:basic-mixin when their titles change.

- (**send** *a-group:* frame-label-changed *frame*) ⇒ void
  frame : (implements frame:basic<%>)

  Updates the windows menu of each frame in the group.

frame-shown/hidden

This method is called by instances of frame:basic% to notify the frame group that a frame's visiblity is changed.

- (**send** *a-group:* frame-shown/hidden) ⇒ void

  Updates the Windows menus of all of the frames in the frame group.

get-active-frame

Returns the frame with the keyboard focus or the first frame in the group.

- (**send** *a-group:* get-active-frame) ⇒ (implements frame:basic<%>)

get-frames

Returns the frames in the group.

- (**send** *a-group:* get-frames) ⇒ (list-of (instance frame:basic<%>))

get-mdi-parent

The result of this method must be used as the parent frame for each frame in the group.

- (**send** *a-group:* get-mdi-parent) ⇒ (union #f (instance frame%))

get-open-here-frame

Returns the currently saved frame to load new files into.

- (**send** *a-group:* get-open-here-frame) ⇒ (union #f (is-a?/c frame:editor<%>))

insert-frame

Inserts a frame into the group.

- (**send** *a-group:* `insert-frame` *frame*) ⇒ void
  frame : (implements `frame:basic<%>`)

`locate-file`

Returns the frame that is editing or viewing a particular file.

- (**send** *a-group:* `locate-file`) ⇒ (union #f (implements `frame:basic<%>`))

`on-close-all`

Call this method to close all of the frames in the group. The function `can-close-all?` must have been called just before this function and it must have returned `#t`.

- (**send** *a-group:* `on-close-all`) ⇒ void
  Calls the `on-close` method and the `show` method (with `#f` as argument) on each frame in the group.

`remove-frame`

Removes a frame from the group.

- (**send** *a-group:* `remove-frame` *frame*) ⇒ void
  frame : (implements `frame:basic<%>`)

`set-active-frame`

Sets the active frame in the group. This method is called by `on-focus`.

- (**send** *a-group:* `set-active-frame` *frame*) ⇒ void
  frame : (implements `frame:basic<%>`)

`set-empty-callbacks`

Sets the empty callbacks. These functions are called when the frame group is empty.

- (**send** *a-group:* `set-empty-callbacks` *test close-down*) ⇒ void
  test : (-¿ boolean)
  close-down : (-¿ void)
  The function *test* is called when there is one frame left in the group. If it returns `#t`, the closing operation may be completed. If it returns `#f`, the closing operation is aborted.
  The function *close-down* is called when there are no frames left in the group.

`set-open-here-frame`

Sets the frame to load new files into. See also `frame:open-here<%>`.

- (**send** *a-group:* `set-open-here-frame` *frame*) ⇒ void
  frame : (is-a?/c `frame:editor%`)

13. Handler

13.0.0.1 Opening a File and Selecting a Format Handler

The function `handler:edit-file` takes a filename and dispatches it to an appropriate *format handler*. A format handler takes a filename and opens a frame for the user to view or edit the file. If no handler is found for a particular format, then the file is opened as a raw text file, in a `frame:text-info-file%` object.

The function `handler:open-file` lets the user select a filename using `finder:get-file`, and then passes the name to `handler:edit-file`.

14. Icon

15. Keymap

- keymap:aug-keymap
- keymap:aug-keymap¡
- keymap:aug-keymap-mixin

15.1 keymap:aug-keymap<%>

Extends: `(class->interface keymap%)`

This keymap overrides some of the built in `keymap%` methods to be able to extract the keybindings from the keymap.

`get-chained-keymaps`

- **(send** *a-keymap:aug-keymap* `get-chained-keymaps`**)** ⇒ (listof (instance `keymap%`))

  Returns the list of keymaps that are chained to this one.

`get-map-function-table`

- **(send** *a-keymap:aug-keymap* `get-map-function-table`**)** ⇒ hash-table

  Returns a hash-table that maps symbols naming key sequences to the names of the keymap functions the are bound to.

`get-map-function-table/ht`

- **(send** *a-keymap:aug-keymap* `get-map-function-table/ht` *ht***)** ⇒ hash-table
  ht : hash-table

  This is a helper function for `get-map-function-table` that returns the same result, except it accepts a hash-table that it inserts the bindings into. It does not replace any bindings already in *ht*.

15.2 keymap:aug-keymap-mixin

Domain: `(class->interface keymap%)`

Implements: `keymap:aug-keymap<%>`

- (make-object keymap:aug-keymap-mixin%) ⇒ keymap:aug-keymap-mixin% object

  Creates an empty keymap.

chain-to-keymap

Multiple keymaps can be chained off one keymap using chain-to-keymap. When keymaps are chained to a main keymap, then events handled by the main keymap are passed to the chained keymaps until a chained keymap handles the events. Keymaps can be chained together in an arbitrary acyclic graph.

Keymap chaining is useful because multiple-event sequences are handled correctly by chained groups. Dispatching each individual event to separate keymaps is problematic without chaining because keymaps may acquire state that must be reset when a callback is invoked in one of the keymaps. This state can be manually cleared with break-sequence, but this also invokes the handler installed by set-break-sequence-callback.

- (**send** *a-keymap:aug-keymap-mixin* chain-to-keymap *next prefix?*) ⇒ void
  next : (instance keymap%)
  prefix? : boolean

  If *next* will be used to handle events which are not handled by this keymap. If *prefix?* is a true value, then *next* will take precedence over other keymaps already chained to this one.

  Keeps a list of the keymaps chained to this one.

map-function

Maps an input state to the name of an event handler.

- (**send** *a-keymap:aug-keymap-mixin* map-function *key-name function-name*) ⇒ void
  key-name : string
  function-name : string

  Maps an input state sequence to a function name using a string-encoded sequence in *keyname*. The format of *keyname* is a sequence of semicolon-delimited input states; each state is made up of a sequence of modifier identifiers followed by a key identifier.

  The modifier identifiers are:

  – "s:" — All platforms: Shift
  – "c:" — All platforms: Control
  – "a:" — Mac OS: Option
  – "m:" — Windows: Alt; X: Meta
  – "d:" — Mac OS: Command

  If a particular modifier is not mentioned in a state string, it matches states whether that modifier is pressed or not pressed. A tilde ( ) preceding a modifier makes the string match only states where the corresponding modifier is not pressed. If the state string begins with a colon, then the string only matches a state if modifiers not mentioned in the string are not pressed.

  A key identifier can be either a character on the keyboard (e.g., "a", "2", "?") or a special name. The special names are:

  – "leftbutton" (button down)
  – "rightbutton"
  – "middlebutton"
  – "leftbuttondouble" (button down for double-click)
  – "rightbuttondouble"
  – "middlebuttondouble"

- "leftbuttontriple" (button down for triple-click)
- "rightbuttontriple"
- "middlebuttontriple"
- "leftbuttonseq" (all events from button down through button up)
- "rightbuttonseq"
- "middlebuttonseq"
- "wheelup"
- "wheeldown"
- "esc"
- "delete"
- "del" (same as "delete")
- "insert"
- "ins" (same as "insert")
- "add"
- "subtract"
- "multiply"
- "divide"
- "backspace"
- "back"
- "return"
- "enter" (same as "return")
- "tab"
- "space"
- "right"
- "left"
- "up"
- "down"
- "home"
- "end"
- "pageup"
- "pagedown"
- "semicolon"
- "colon"
- "numpad1"
- "numpad2"
- "numpad3"
- "numpad4"
- "numpad5"
- "numpad6"
- "numpad7"
- "numpad8"
- "numpad9"
- "f1"
- "f2"
- "f3"
- "f4"
- "f5"
- "f6"
- "f7"
- "f8"
- "f9"
- "f10"
- "f11"

- "f12"
- "f13"
- "f14"
- "f15"
- "f16"
- "f17"
- "f18"
- "f19"
- "f20"
- "f21"
- "f22"
- "f23"
- "f24"

For a special keyword, the capitalization does not matter. However, capitalization is important for single-letter keynames (e.g., "A" is interpreted as "s:a").

A state can match multiple state strings mapped in a keymap (or keymap chain); when a state matches multiple state strings, a mapping is selected by ranking the strings according to specificity. A state string that mentions more pressed modifiers has a higher rank than other state strings, and if two strings mention the same number of pressed modifiers, the one that metions more unpressed modifiers has a higher rank. In that case that multiple matching strings have the same rank, one string is selected arbitrarily.

Examples:

- "space" — matches whenever the space bar is pressed, regardless of the state of modifiers keys.
- " c:space" — matches whenever the space bar is pressed and the Control key is not pressed.
- "a" — matches whenever "a" is typed, regardless of the state of modifiers keys other than Shift.
- ":a" — matches only when "a" is typed with no modifier keys pressed.
- " c:a" — matches whenever "a" is typed and neither the Shift key nor the Control key is pressed.
- ":esc;:c:c" — matches an Escape key press (no modifiers) followed by a Control-C press (no modifiers other than Control).

A call to map-function that would map a particular key sequence both as a prefix and as a complete sequence raises an exception, but the exception handler cannot escape (see Exceptions and Continuation Jumps, §2.4.4 in *PLT MrEd: Graphical Toolbox Manual*).

A function name does not have to be mapped to a handler before input states are mapped to the name; the handler is dispatched by name at the time of invocation. The event handler mapped to a function name can be changed without affecting the map from input states to function names.

Keeps a separate record of the key names and functions that they are bound to in this keymap.

15.3 keymap:aug-keymap% = (keymap:aug-keymap-mixin keymap%)

keymap:aug-keymap% = (keymap:aug-keymap-mixin keymap%)

- (make-object keymap:aug-keymap%) ⇒ keymap:aug-keymap% object

  Creates an empty keymap.

16. Main

17. Match Cache

- match-cache:

17.1 match-cache:%

This class defines a cache that can be used with paren:backward-match and paren:forward-match. The cache data is intended to be inserted and read only by the matching procedures; however, the cache must be specifically invalidated when the cache's buffer is modified.

A cache is not required by the matching procedures. A single cache can only be used for a single buffer and with a single direction and parenthesis, quote, and comment parameterization. When a cache is used for a buffer, it does not have to be used with every call to the matching procedures, but the best results are obtained when the cache is always used. Multiple caches can be used for a single buffer (possibly for different directions or parenthesis parameterizations), as long as they are all invalidated properly when the buffer is modified.

A text's after-insert and after-delete methods can be overridden to properly track modifications and invoke a cache's invalidate or forward-invalidate methods.

`contents`

UNDOCUMENTED

- (**send** *a-match-cache:* `contents`) ⇒ void

`delete`

UNDOCUMENTED

- (**send** *a-match-cache:* `delete`) ⇒ void

`forward-invalidate`

- (**send** *a-match-cache:* `forward-invalidate` *pos*) ⇒ void
  pos : integer

  Call this method when text is inserted or deleted at position *pos* in the cache's buffer and the cache is used for forward-matching.

`get`

UNDOCUMENTED

- (**send** *a-match-cache:* `get`) ⇒ void

`invalidate`

- (**send** *a-match-cache:* `invalidate` *pos*) ⇒ void
  pos : integer

  Call this method when text is inserted or deleted at position *pos* in the cache's buffer and the cache is used for backward-matching.

`put`

UNDOCUMENTED

- (**send** *a-match-cache:* `put`) ⇒ void

`splay`

UNDOCUMENTED

- (**send** *a-match-cache:* `splay`) ⇒ void

18. Menu

- menu:can-restore-checkable-menu-item
- menu:can-restore-menu-item
- menu:can-restore-underscore-menu
- menu:can-restore-underscore¡
- menu:can-restore¡
- menu:can-restore-mixin
- menu:can-restore-underscore-mixin

18.1 menu:can-restore<%>

Extends: selectable-menu-item<%>

Classes created with this mixin remember their keybindings so the keybindings can be removed and then restored.

`restore-keybinding`

Sets the keyboard shortcut to the setting it had when the class was created.

- (**send** *a-menu:can-restore* `restore-keybinding`) ⇒ void

18.2 menu:can-restore-mixin

Domain: selectable-menu-item<%>

Implements: selectable-menu-item<%>

Implements: menu:can-restore<%>

18.3 menu:can-restore-underscore<%>

Extends: labelled-menu-item<%>

These menus can save and restore the underscores (indicated via the & characters in the original labels) in their labels.

If the preference 'framework:menu-bindings is \#f, calls erase-underscores during initialization.

erase-underscores

Erases the underscores in the label of this menu, but remebers them so they can be restores with restore-underscores.

- (**send** *a-menu:can-restore-underscore* erase-underscores) ⇒ void

restore-underscores

Restores underscores in the menu's label to their original state.

- (**send** *a-menu:can-restore-underscore* restore-underscores) ⇒ void

18.4 menu:can-restore-underscore-mixin

Domain: labelled-menu-item<%>

Implements: labelled-menu-item<%>

Implements: menu:can-restore-underscore<%>

18.5 menu:can-restore-menu-item% = (menu:can-restore-mixin menu-item%)

menu:can-restore-menu-item% = (menu:can-restore-mixin menu-item%)

- (**instantiate** menu:can-restore-menu-item% () (**label** _) (**parent** _) (**callback** _) [(**shortcut** _)] [(**help-string** _)] [(**demand-callback** _)]) ⇒ menu:can-restore-menu-item% object
  label : string (up to 200 characters)
  parent : menu% or popup-menu% object
  callback : procedure of two arguments: a menu-item% object and a control-event% object
  shortcut = #f : character or #f
  help-string = #f : string (up to 200 characters) or #f
  demand-callback = void : procedure of one argument: a menu-item% object

  Creates a new menu item in *parent*. The item is initially shown, appended to the end of its parent. The *callback* procedure is called (with the event type 'menu) when the user selects the menu item (either via a menu bar, popup-menu in window<%>, or popup-menu in editor-admin%).

  See set-label for information about mnemonic ampersands ("&") in *label*.

  If shortcut is not #f, the item has a shortcut. See get-shortcut for more information.

  If *help-string* is not #f, the item has a help string. See get-help-string for more information.

  The *demand-callback* procedure is called by the default on-demand method with the object itself.

18.6 menu:can-restore-checkable-menu-item% = (menu:can-restore-mixin checkable-menu-ite

menu:can-restore-checkable-menu-item% = (menu:can-restore-mixin checkable-menu-item%)

- **(instantiate** menu:can-restore-checkable-menu-item% () **(label** _) **(parent** _) **(callback** _) **[(shortcut** _)] **[(help-string** _)] **[(demand-callback** _)]) ⇒ menu:can-restore-checkable-menu-item% object
  label : string (up to 200 characters)
  parent : menu% or popup-menu% object
  callback : procedure of two arguments: a menu-item% object and a control-event% object
  shortcut = #f : character or #f
  help-string = #f : string (up to 200 characters) or #f
  demand-callback = void : procedure of one argument: a checkable-menu-item% object

  Creates a new menu item in *parent*. The item is initially shown, appended to the end of its parent, and unchecked. The *callback* procedure is called (with the event type 'menu) when the menu item is selected (either via a menu bar, popup-menu in window<%>, or popup-menu in editor-admin%).

  See set-label for information about mnemonic ampersands ("&") in *label*.

  If shortcut is not #f, the item has a shortcut. See get-shortcut for more information.

  If *help-string* is not #f, the item has a help string. See get-help-string for more information.

  The *demand-callback* procedure is called by the default on-demand method with the object itself.

18.7 menu:can-restore-underscore-menu% = (menu:can-restore-underscore-mixin menu%)

menu:can-restore-underscore-menu% = (menu:can-restore-underscore-mixin menu%)

- **(instantiate** menu:can-restore-underscore-menu% () **(label** _) **(parent** _) **[(help-string** _)] **[(demand-callback** _)]) ⇒ menu:can-restore-underscore-menu% object
  label : string (up to 200 characters)
  parent : menu%, popup-menu%, or menu-bar% object
  help-string = #f : string (up to 200 characters) or #f
  demand-callback = void : procedure of one argument: a menu% object

  Creates a new menu with the given label.

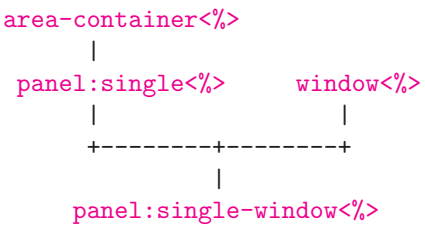
  If *label* contains an ampersand ("&"), it is handled specially; under Windows, the character following an ampersand is underlined in the displayed menu title to indicate a keyboard mnemonic. Pressing and releasing the Alt key switches to menu-selection mode in the menu bar where mnemonic characters are used for navigation. An Alt combination might select a specific menu via on-menu-char. A double-ampersand in *label* is replaced by a literal (non-navigation) ampersand. Under X and Mac OS, ampersands in the label are parsed in the same way as for Windows, but no mnemonic underline is displayed.

  If *help-string* is not #f, the menu has a help string. See get-help-string for more information.

  The *demand-callback* procedure is called by the default on-demand method with the object itself.

  If the menu has the label "Help" in a menu bar, it is treated specially on some platforms. Under Mac OS, the items of a "Help" menu are folded into the standard help menu. In addition, under Mac OS, if the name of the first item in the "Help" menu starts with "About", then the menu item is duplicated as the first item under the Apple menu.

19. Panel



- panel:horizontal-dragable
- panel:single-pane
- panel:single
- panel:vertical-dragable
- panel:dragable¡
- panel:horizontal-dragable¡
- panel:single-window¡
- panel:single¡
- panel:vertical-dragable¡
- panel:dragable-mixin
- panel:horizontal-dragable-mixin
- panel:single-mixin
- panel:single-window-mixin
- panel:vertical-dragable-mixin

19.1 panel:single<%>

Extends: area-container<%>

See panel:single-mixin%.

`active-child`

- (**send** *a-panel:single* `active-child` *child*) ⇒ void
  child : (implements `area<%>`)

  Sets the active child to be *child*

- (**send** *a-panel:single* `active-child`) ⇒ (implements `area<%>`)

  Returns the current active child.

19.2 panel:single-mixin

Domain: `area-container<%>`

Implements: `area-container<%>`

Implements: `panel:single<%>`

This mixin adds single panel functionality to an implementation of the `area-container<%>` interface.

Single panels place all of the children in the center of the panel, and allow make one child to be visible at a time. The `active-child` method controls which panel is currently active.

The `show` method is used to hide and show the children of a single panel.

`after-new-child`

This method is called after a new containee area is created with this area as its container. The new child is provided as an argument to the method.

- (**send** *a-panel:single-mixin* `after-new-child` *child*) ⇒ void
  child : `subarea<%>`

  Does nothing.

  Hides this child by calling

  ```
  (send child show #f)
  ```

  , unless this is the first child in which case it does nothing.

`container-size`

Called to determine the minimum size of a container. See Geometry Management, §2.2 in *PLT MrEd: Graphical Toolbox Manual* for more information.

- (**send** *a-panel:single-mixin* `container-size`) ⇒ (values exact-integer exact-integer)

  Returns the maximum width of all the children and the maximum height of all of the children.

`place-children`

Called to place the children of a container. See Geometry Management, §2.2 in *PLT MrEd: Graphical Toolbox Manual* for more information.

- (**send** *a-panel:single-mixin* `place-children`) ⇒ (listof (list exact-integer exact-integer exact-integer exact-integer))

  Returns the positions for single panels and panes.

19.3 panel:single-window<%>

Extends: window<%>

Extends: panel:single<%>

19.4 panel:single-window-mixin

Domain: window<%>

Domain: panel:single<%>

Implements: panel:single<%>

Implements: window<%>

Implements: panel:single-window<%>

`container-size`

Called to determine the minimum size of a container. See Geometry Management, §2.2 in *PLT MrEd: Graphical Toolbox Manual* for more information.

- (**send** *a-panel:single-window-mixin* `container-size` *info*) ⇒ (values exact-integer exact-integer)
  info : (list-of (list exact-integer exact-integer boolean boolean))

  Factors the border width into the size calculation.

19.5 panel:single% = (panel:single-mixin (panel:single-window-mixin panel%))

```
panel:single% = (panel:single-mixin (panel:single-window-mixin panel%))
```

- (**instantiate** `panel:single%` () (**parent** _) [(**style** _)] [(**enabled** _)] [(**vert-margin** _)] [(**horiz-margin** _)] [(**border** _)] [(**spacing** _)] [(**alignment** _)] [(**min-width** _)] [(**min-height** _)]

[(**stretchable-width** _)] [(**stretchable-height** _)]) ⇒ panel:single% object
parent : frame%, dialog%, panel%, or pane% object
style = null : list of symbols in '(border)
enabled = #t : boolean
vert-margin = 0 : exact integer in [0, 1000]
horiz-margin = 0 : exact integer in [0, 1000]
border = 0 : exact integer in [0, 1000]
spacing = 0 : exact integer in [0, 1000]
alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
min-width = 0 : exact integer in [0, 10000]
min-height = 0 : exact integer in [0, 10000]
stretchable-width = #t : boolean
stretchable-height = #t : boolean

If the 'border style is specified, the window is created with a thin border (only in this case, the client size of the panel may be less than its total size).

For information about the *enabled* argument, see window<%>. For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

19.6 panel:single-pane% = (panel:single-mixin pane%)

panel:single-pane% = (panel:single-mixin pane%)

- (**instantiate** panel:single-pane% () (**parent** _) [(**vert-margin** _)] [(**horiz-margin** _)] [(**border** _)] [(**spacing** _)] [(**alignment** _)] [(**min-width** _)] [(**min-height** _)] [(**stretchable-width** _)] [(**stretchable-height** _)]) ⇒ panel:single-pane% object
  parent : frame%, dialog%, panel%, or pane% object
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

19.7 panel:dragable<%>

Classes matching this interface implement a panel where the user can adjust the percentage of the space that each takes up. The user adjusts the size by clicking an dragging the empty space between the children.

after-percentage-change

This method is called when the user changes the percentage by dragging the bar between the children, or when a new child is added to the frame, but not when set-percentages is called.

- (**send** *a-panel:dragable* `after-percentage-change` *new-percentage*) ⇒ void
  new-percentage : number

`get-percentages`

Return the current percentages of the children.

- (**send** *a-panel:dragable* `get-percentages`) ⇒ (listof numbers)

`get-vertical?`

This method controls the behavior of the other overridden methods in mixins that implement this interface.

If it returns #t, the panel will be vertically aligned and if it returns #f, they will be horizontally aligned.

- (**send** *a-panel:dragable* `get-vertical?`) ⇒ boolean

`set-percentages`

Call this method to set the percentages that each window takes up of the panel.

- (**send** *a-panel:dragable* `set-percentages` *new-percentages*) ⇒ void
  new-percentages : (listof number)

  The argument, *new-percentages* must be a list of numbers that sums to 1. It's length must be equal to the number of children of the panel (see `get-children`) and each percentage must correspond to a number of pixels that is equal to or larger than the minimum with of the child, as reported by `min-width`.

19.8 panel:vertical-dragable<%>

Extends: `panel:dragable<%>`

A panel that implements `panel:vertical-dragable<%>`. It aligns its children vertically.

19.9 panel:horizontal-dragable<%>

Extends: `panel:dragable<%>`

A panel that implements `panel:horizontal-dragable<%>`. It aligns its children horizontally.

19.10 panel:dragable-mixin

Domain: `(class->interface panel%)`

Implements: `panel:dragable<%>`

This mixin adds the panel:dragable<%> functionality to a panel%.

It is not useful unless the get-vertical? method is overridden.

- **(instantiate** panel:dragable-mixin% () **(parent** _) [**(style** _)] [**(enabled** _)] [**(vert-margin** _)] [**(horiz-margin** _)] [**(border** _)] [**(spacing** _)] [**(alignment** _)] [**(min-width** _)] [**(min-height** _)] [**(stretchable-width** _)] [**(stretchable-height** _)]) ⇒ panel:dragable-mixin% object
  parent : frame%, dialog%, panel%, or pane% object
  style = null : list of symbols in '(border)
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  If the 'border style is specified, the window is created with a thin border (only in this case, the client size of the panel may be less than its total size).

  For information about the *enabled* argument, see window<%>. For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

after-new-child

This method is called after a new containee area is created with this area as its container. The new child is provided as an argument to the method.

- **(send** *a-panel:dragable-mixin* after-new-child *child*) ⇒ void
  child : (instance-of (implements area<%>))

  Updates the number of percentages to make sure that it matches the number of children and calls after-percentage-change.

on-subwindow-event

Called when this window or a child window receives a mouse event. The on-subwindow-event method of the receiver's top-level window is called first (see get-top-level-window); if the return value is #f, the on-subwindow-event method is called for the next child in the path to the receiver, and so on. Finally, if the receiver's on-subwindow-event method returns #f, the event is passed on to the receiver's normal mouse-handling mechanism.

- **(send** *a-panel:dragable-mixin* on-subwindow-event *receiver event*) ⇒ boolean
  receiver : (instanceof window<%>)
  event : (instanceof mouse-event%)

  The *event* argument is the event that was generated for the *receiver* window. Returns #f.

  When the cursor is dragging the middle bar around, this method handles the resizing of the two panes.

`place-children`

Called to place the children of a container. See Geometry Management, §2.2 in *PLT MrEd: Graphical Toolbox Manual* for more information.

- (**send** *a-panel:dragable-mixin* `place-children` *info w h*) ⇒ (list-of (list exact-int exact-int exact-int exact-int))
  info : (list-of (list exact-int exact-int))
  w : exact-int
  h : exact-int

  Places the children vertically in the panel, based on the percentages returned from `get-percentages`. Also leaves a little gap between each pair of children.

19.11 panel:vertical-dragable-mixin

Domain: `panel:dragable<%>`

Implements: `panel:dragable<%>`

Implements: `panel:vertical-dragable<%>`

This mixin merely overrides the `get-vertical?` method of the `panel:dragable-mixin` to return #t.

`get-vertical?`

This method controls the behavior of the other overridden methods in mixins that implement this interface.

If it returns #t, the panel will be vertically aligned and if it returns #f, they will be horizontally aligned.

- (**send** *a-panel:vertical-dragable-mixin* `get-vertical?`) ⇒ boolean
  Returns #t.

19.12 panel:horizontal-dragable-mixin

Domain: `panel:dragable<%>`

Implements: `panel:dragable<%>`

Implements: `panel:vertical-dragable<%>`

This mixin merely overrides the `get-vertical?` method of the `panel:dragable-mixin` to return #f.

`get-vertical?`

This method controls the behavior of the other overridden methods in mixins that implement this interface.

If it returns #t, the panel will be vertically aligned and if it returns #f, they will be horizontally aligned.

- **(send** *a-panel:horizontal-dragable-mixin* get-vertical?) ⇒ boolean
  Returns #f.

19.13 panel:vertical-dragable% = (panel:dragable-mixin (panel:vertical-dragable-mixin vertical-panel%))

```
panel:vertical-dragable% = (panel:dragable-mixin (panel:vertical-dragable-mixin vertical-panel%))
```

- **(instantiate** panel:vertical-dragable% () **(parent** _) [**(style** _)] [**(enabled** _)] [**(vert-margin** _)] [**(horiz-margin** _)] [**(border** _)] [**(spacing** _)] [**(alignment** _)] [**(min-width** _)] [**(min-height** _)] [**(stretchable-width** _)] [**(stretchable-height** _)]) ⇒ panel:vertical-dragable% object
  parent : frame%, dialog%, panel%, or pane% object
  style = null : list of symbols in '(border)
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean
  If the 'border style is specified, the window is created with a thin border (only in this case, the client size of the panel may be less than its total size).
  For information about the *enabled* argument, see window<%>. For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

- **(instantiate** panel:vertical-dragable% () **(parent** _) [**(style** _)] [**(enabled** _)] [**(vert-margin** _)] [**(horiz-margin** _)] [**(border** _)] [**(spacing** _)] [**(alignment** _)] [**(min-width** _)] [**(min-height** _)] [**(stretchable-width** _)] [**(stretchable-height** _)]) ⇒ panel:vertical-dragable% object
  parent : frame%, dialog%, panel%, or pane% object
  style = null : list of symbols in '(border)
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean
  If the 'border style is specified, the window is created with a thin border (only in this case, the client size of the panel may be less than its total size).

For information about the *enabled* argument, see window<%>. For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

19.14 panel:horizontal-dragable% = (panel:horizontal-dragable-mixin (panel:dragable-mix horizontal-panel%))

```
panel:horizontal-dragable% = (panel:horizontal-dragable-mixin (panel:dragable-mixin horizontal-panel%))
```

- **(instantiate** panel:horizontal-dragable% **()** **(parent** _**)** **[(style** _**)]** **[(enabled** _**)]** **[(vert-margin** _**)]** **[(horiz-margin** _**)]** **[(border** _**)]** **[(spacing** _**)]** **[(alignment** _**)]** **[(min-width** _**)]** **[(min-height** _**)]** **[(stretchable-width** _**)]** **[(stretchable-height** _**)])** ⇒ panel:horizontal-dragable% object
  parent : frame%, dialog%, panel%, or pane% object
  style = null : list of symbols in '(border)
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  If the 'border style is specified, the window is created with a thin border (only in this case, the client size of the panel may be less than its total size).

  For information about the *enabled* argument, see window<%>. For information about the *horiz-margin* and *vert-margin* arguments, see subarea<%>. For information about the *border*, *spacing*, and *alignment* arguments, see area-container<%>. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see area<%>.

- **(instantiate** panel:horizontal-dragable% **()** **(parent** _**)** **[(style** _**)]** **[(enabled** _**)]** **[(vert-margin** _**)]** **[(horiz-margin** _**)]** **[(border** _**)]** **[(spacing** _**)]** **[(alignment** _**)]** **[(min-width** _**)]** **[(min-height** _**)]** **[(stretchable-width** _**)]** **[(stretchable-height** _**)])** ⇒ panel:horizontal-dragable% object
  parent : frame%, dialog%, panel%, or pane% object
  style = null : list of symbols in '(border)
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  border = 0 : exact integer in [0, 1000]
  spacing = 0 : exact integer in [0, 1000]
  alignment = '(left top) : two-element list: 'left, 'center or 'right and 'top, 'center or 'bottom
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  If the 'border style is specified, the window is created with a thin border (only in this case, the client size of the panel may be less than its total size).

For information about the *enabled* argument, see `window<%>`. For information about the *horiz-margin* and *vert-margin* arguments, see `subarea<%>`. For information about the *border*, *spacing*, and *alignment* arguments, see `area-container<%>`. For information about the *min-width*, *min-height*, *stretchable-width*, and *stretchable-height* arguments, see `area<%>`.

20. Parenthesis

MrEd provides general-purpose "parenthesis"-matching utilities that work on buffers. The utilities are paramerized with respect to:

- *parens* — Pairs (`cons` cells) of opening and closing bracket strings. These brackets can be nested and must be balanced. The opening and closing string do not have to be different.
- *quotes* — Pairs of opening and closing quote bracket strings. Within a pair of quotes, all other bracket, quote, and comment strings are ignored. Pairs of quote strings do not have to correspond to quotes in the language. For example, C comments using "/*" and "*/" are considered quotes for parenthesis-matching purposes.
- *comments* — A list of comment-starting strings. Comments of this form run until the end of the line.

A backslash (\) is assumed to be the (only) method for escaping parenthesis, quote, and comment markers. The parenthesis-balancing utilities are only guaranteed to act meaningfully when the starting position is outside quoted and commented expressions. When a pair of opening and closing brackets are not distinct, the actions on these brackets are only meaningful for searches starting outside the brackets.

21. Pasteboard

- pasteboard:backup-autosave
- pasteboard:basic
- pasteboard:file
- pasteboard:info
- pasteboard:keymap

21.1 pasteboard:basic% = (editor:basic-mixin pasteboard%)

```
pasteboard:basic% = (editor:basic-mixin pasteboard%)
```

- **(instantiate** pasteboard:basic% **())** ⇒ pasteboard:basic% object
  The editor will not be displayed until it is attached to a editor-canvas% object or some other display.
  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.
  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

21.2 pasteboard:keymap% = (editor:keymap-mixin pasteboard:basic%)

```
pasteboard:keymap% = (editor:keymap-mixin pasteboard:basic%)
```

- **(instantiate** pasteboard:keymap% **())** ⇒ pasteboard:keymap% object
  The editor will not be displayed until it is attached to a editor-canvas% object or some other display.
  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.
  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

21.3 pasteboard:file% = (editor:file-mixin pasteboard:keymap%)

```
pasteboard:file% = (editor:file-mixin pasteboard:keymap%)
```

- **(instantiate** pasteboard:file% **())** ⇒ pasteboard:file% object
  The editor will not be displayed until it is attached to a editor-canvas% object or some other display.
  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.
  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

21.4 pasteboard:backup-autosave% = (editor:backup-autosave-mixin pasteboard:file%)

pasteboard:backup-autosave% = (editor:backup-autosave-mixin pasteboard:file%)

- (**instantiate** pasteboard:backup-autosave% ()) ⇒ pasteboard:backup-autosave% object

  The editor will not be displayed until it is attached to a editor-canvas% object or some other display.

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

21.5 pasteboard:info% = (editor:info-mixin pasteboard:backup-autosave%)

pasteboard:info% = (editor:info-mixin pasteboard:backup-autosave%)

- (**instantiate** pasteboard:info% ()) ⇒ pasteboard:info% object

  The editor will not be displayed until it is attached to a editor-canvas% object or some other display.

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

22. Pathname Utilities

23. Preferences

The framework provides a user preferences manager. It provides facilities for getting, setting, marshalling and unmarshalling the user's preferences as well as utilities to manage a preferences dialog box.

In addition to the functions `preferences:add-scheme-checkbox-panel`, `preferences:add-warnings-checkbox-panel`, `preferences:add-editor-checkbox-panel`, and `preferences:add-font-panel` listed here, `scheme:add-preferences-pan` also adds panels to the preferences dialog.

24. Scheme

- scheme:sexp-snip
- scheme:text
- scheme:sexp-snip¡
- scheme:text¡
- scheme:text-mixin

24.1 scheme:sexp-snip<%>

`get-saved-snips`

This returns the list of snips hidden by the sexp snip.

- (**send** *a-scheme:sexp-snip* `get-saved-snips`) ⇒ (listof `snip%`)

24.2 scheme:sexp-snip%

Superclass: `snip%`

Implements: `readable-snip<%>`

Implements: `scheme:sexp-snip<%>`

- (`make-object scheme:sexp-snip%`) ⇒ `scheme:sexp-snip%` object

  Creates a plain snip of length 1.

`copy`

Creates and returns a copy of this snip. The `copy` method is responsible for copying this snip's style (as returned by `get-style`) to the new snip.

- (**send** *a-scheme:sexp-snip* `copy`) ⇒ (is-a?/c `scheme:sexp-snip%`)

  Returns a copy of this snip that includes the hidden snips.

`draw`

Called (by an editor) to draw the snip.

Before this method is called, the correct font, text color, and pen color will have been set in the drawing context for this snip already. (This is *not* true for `get-extent` or `partial-offset`.) The `draw` method must not make any other assumptions about the state of the drawing context, except that the clipping region is already set to something appropriate. Before `draw` returns, it must restore any drawing context settings that it changes.

See also `on-paint` in `editor<%>`.

The snip's editor is usually internally locked for writing and reflowing when this method is called (see also "Locks" (section 8.8, page 169)).

- (**send** *a-scheme:sexp-snip* `draw` *dc x y left top right bottom dx dy draw-caret*) ⇒ void
  dc : `dc<%>` object
  x : real number
  y : real number
  left : real number
  top : real number
  right : real number
  bottom : real number
  dx : real number
  dy : real number
  draw-caret : symbol in '(no-caret show-inactive-caret show-caret)

  Draws brackets with a centered ellipses between them.

`get-extent`

Calculates the snip's width, height, descent (amount of height which is drawn below the baseline), space (amount of height which is "filler" space at the top), and horizontal spaces (amount of width which is "filler" space at the left and right).

This method is called by the snip's administrator; it should not be called directly by others. To get the extent of a snip, use `get-snip-location` in `editor<%>` .

A drawing context is provided for the purpose of finding font sizes, but no drawing should occur. The `get-extent` and `partial-offset` methods must not make any assumptions about the state of the drawing context, except that it is scaled properly. In particular, the font for the snip's style is not automatically set in the drawing context before the method is called.[1] If `get-extent` or `partial-offset` changes the drawing context's setting, it must restore them before returning. However, the methods should not need to change the drawing context; only font settings can affect measurement results from a device context, and `get-text-extent` in `dc<%>` accepts a `font%` argument for sizing that overrides that device context's current font.

The snip's left and top locations are provided in editor coordinates. In a text editor, the y-coordinate is the *line's* top location; the snip's actual top location is potentially undetermined until its height is known.

If a snip caches the result size for future replies, it should invalidate its cached size when `size-cache-invalid` is called (especially if the snip's size depends on any device context properties).

If a snip's size changes after receiving a call to `get-extent` and before receiving a call to

[1] Many snips cache their size information, so automatically setting the font would be wasteful.

`size-cache-invalid`, then the snip must notify its administrator of the size change, so that the administrator can recompute its derived size information. Notify the administrator of a size change by call its `resized` method.

The snip's editor is usually internally locked for writing and reflowing when this method is called (see also "Locks" (section 8.8, page 169)).

- (**send** *a-scheme:sexp-snip* `get-extent` *dc x y w h descent space lspace rspace*) ⇒ void
  dc : (is-a?/c `dc<%>`)
  x : real number
  y : real number
  w = #f : boxed non-negative real number or #f
  h = #f : boxed non-negative real number or #f
  descent = #f : boxed non-negative real number or #f
  space = #f : boxed non-negative real number or #f
  lspace = #f : boxed non-negative real number or #f
  rspace = #f : boxed non-negative real number or #f

  Returns a size corresponding to what this snip draws.

`get-text`

Gets the text representation for this snip.

- (**send** *a-scheme:sexp-snip* `get-text` *offset num flattened?*) ⇒ string
  offset : number
  num : number
  flattened? = #f : boolean

  Returns the text for this snip starting with the position *offset* within the snip, and continuing for a total length of *num* items. If *offset* is greater than the snip's count, then "" is returned. If *num* is greater than the snip's count minus the offset, then text from the offset to the end of the snip is returned.

  If *flattened?* is not #f, then flattened text is returned. See "Getting Text" (section 8.4, page 168) for a discussion of flattened vs. non-flattened text.

  Returns the concatenation of the text for all of the hidden snips.

`write`

Writes the snip to the given stream. (Snip reading is handled by the snip class.) Style information about the snip (i.e., the content of `get-style`) will be saved and restored automatically.

- (**send** *a-scheme:sexp-snip* `write` *stream-out*) ⇒ void
  stream-out : `editor-stream-out%`

  Saves the embedded snips

24.3 `scheme:text<%>`

Texts matching this interface support Scheme mode operations.

`backward-sexp`

Move the caret backwards one sexpression

- **(send** *a-scheme:text* `backward-sexp` *start-pos***)** ⇒ void
  start-pos : exact-integer

  Moves the caret to the beginning of the sexpression that ends at *start-pos*.

`balance-parens`

This function is called when the user types a close parenthesis in the `text%`. If the close parenthesis that the user inserted does not match the corresponding open parenthesis and the `'framework:fixup-parens` preference is `#t` (see `preferences:get`) the correct closing parenthesis is inserted. If the `'framework:paren-match` preference is `#t` (see `preferences:get`) the matching open parenthesis is flashed.

- **(send** *a-scheme:text* `balance-parens` *key-event***)** ⇒ void
  key-event : (instance `key-event%`)

`balance-quotes`

This function is called when the user types a quote in the `text%`. If the `'framework:paren-match` preference is `#t` (see `preferences:get`) the matching open quote is flashed.

- **(send** *a-scheme:text* `balance-quotes` *key-event***)** ⇒ void
  key-event : (instance `key-event%`)

`box-comment-out-selection`

This method comments out a selection in the text by putting it into a comment box.

- **(send** *a-scheme:text* `box-comment-out-selection` *start-pos* *end-pos***)** ⇒ void
  start-pos = `'start` : (union 'start exact-integer)
  end-pos = `'end` : (union 'end exact-integer)

  Removes the region from *start-pos* to *end-pos* from the editor and inserts a comment box with that region of text inserted into the box.

  If *start-pos* is `'start`, the starting point of the selection is used. If *end-pos* is `'end`, the ending point of the selection is used.

`comment-out-selection`

- **(send** *a-scheme:text* `comment-out-selection` *start* *end***)** ⇒ void
  start : exact-integer
  end : exact-integer

  Comments the lines containing positions *start* through *end* by inserting a semi-colon at the front of each line.

`down-sexp`

- **(send** *a-scheme:text* `down-sexp` *start***)** ⇒ void
  start : exact-integer

Moves forward into the next S-expression after the position *start*.

`find-down-sexp`

- **(send** *a-scheme:text* `find-down-sexp` *start-pos***)** ⇒ (union #f exact-integer)
  start-pos : exact-integer

  Returns the position of the beginning of the next sexpression inside the sexpression that contains *start-pos*. If there is no such sexpression, it returns `#f`.

`find-up-sexp`

- **(send** *a-scheme:text* `find-up-sexp` *start-pos***)** ⇒ (union #f exact-integer)
  start-pos : exact-integer

  Returns the position of the beginning of the next sexpression outside the sexpression that contains *start-pos*. If there is no such sexpression, it returns `#f`.

`flash-backward-sexp`

- **(send** *a-scheme:text* `flash-backward-sexp` *start-pos***)** ⇒ void
  start-pos : exact-integer

  Flashes the parenthesis that opens the sexpression at *start-pos*.

`flash-forward-sexp`

- **(send** *a-scheme:text* `flash-forward-sexp` *start-pos***)** ⇒ void
  start-pos : exact-integer

  Flashes the parenthesis that closes the sexpression at *start-pos*.

`forward-sexp`

- **(send** *a-scheme:text* `forward-sexp` *start***)** ⇒ exact-integer
  start : #t

  Moves forward over the S-expression starting at position *start*.

`get-backward-sexp`

- **(send** *a-scheme:text* `get-backward-sexp` *start***)** ⇒ (union exact-integer #f)
  start : exact-integer

  Returns the position of the start of the S-expression before or containing *start*, or `#f` if there is no appropriate answer.

`get-forward-sexp`

- **(send** *a-scheme:text* `get-forward-sexp` *start***)** ⇒ (union #f exact-integer)
  start : exact-integer

  Returns the position of the end of next S-expression after position *start*, or `#f` if there is no appropriate answer.

`get-limit`

- **(send** *a-scheme:text* `get-limit` *start*) ⇒ int
  start : exact-integer

  Returns a limit for backward-matching parenthesis starting at position *start*.

`get-tab-size`

This method returns the current size of the tabs for scheme mode. See also `set-tab-size`.

- **(send** *a-scheme:text* `get-tab-size`) ⇒ exact-integer

`highlight-parens`

- **(send** *a-scheme:text* `highlight-parens` *just-clear?*) ⇒ void
  just-clear? = #f : boolean

  When the current position is at the beginning or the end of an sexpression, this method highlights the sexpression.

`insert-return`

- **(send** *a-scheme:text* `insert-return`) ⇒ void

  Inserts a newline into the buffer. If `tabify-on-return?` returns `#t`, this will tabify the new line.

`mark-matching-parenthesis`

If the paren after *pos* is matched, this method higlights it and it's matching counterpart in dark green.

- **(send** *a-scheme:text* `mark-matching-parenthesis` *pos*) ⇒ void
  pos : exact-positive-integer

`remove-parens-forward`

- **(send** *a-scheme:text* `remove-parens-forward` *start*) ⇒ void
  start : exact-integer

  Removes the parentheses from the S-expression starting after the position *start*.

`remove-sexp`

- **(send** *a-scheme:text* `remove-sexp` *start*) ⇒ void
  start : exact-integer

  Forward-deletes the S-expression starting after the position *start*.

`select-backward-sexp`

- **(send** *a-scheme:text* `select-backward-sexp` *start*) ⇒ #t
  start : exact-integer

  Selects the previous S-expression, starting at position *start*.

`select-down-sexp`

- (**send** *a-scheme:text* `select-down-sexp` *start*) ⇒ #t
  start : exact-integer

  Selects the region to the next contained S-expression, starting at position *start*.

`select-forward-sexp`

- (**send** *a-scheme:text* `select-forward-sexp` *start*) ⇒ #t
  start : exact-integer

  Selects the next S-expression, starting at position *start*.

`select-up-sexp`

- (**send** *a-scheme:text* `select-up-sexp` *start*) ⇒ #t
  start : exact-integer

  Selects the region to the enclosing S-expression, starting at position *start*.

`set-tab-size`

This method sets the tab size for this text.

- (**send** *a-scheme:text* `set-tab-size` *new-size*) ⇒ void
  new-size : exact-integer

`tabify`

- (**send** *a-scheme:text* `tabify` *start-pos*) ⇒ void
  start-pos = (send this `get-start-position`) : exact-integer

  Tabs the line containing by *start-pos*

`tabify-all`

- (**send** *a-scheme:text* `tabify-all`) ⇒ void

  Tabs all lines.

`tabify-on-return?`

The result of this method is used to determine if the return key automatically tabs over to the correct position.

Override it to change it's behavior.

- (**send** *a-scheme:text* `tabify-on-return?`) ⇒ boolean

`tabify-selection`

- (**send** *a-scheme:text* `tabify-selection` *start end*) ⇒ void

start : exact-integer
end : exact-integer

Sets the tabbing for the lines containing positions *start* through *end*.

`transpose-sexp`

- (**send** *a-scheme:text* `transpose-sexp` *start*) ⇒ void
  start : exact-integer

  Swaps the S-expression beginning before the position *start* with the next S-expression following *start*.

`uncomment-selection`

- (**send** *a-scheme:text* `uncomment-selection` *start* *end*) ⇒ void
  start : int
  end : int

  Uncomments the lines containing positions *start* through *end*.

`up-sexp`

- (**send** *a-scheme:text* `up-sexp` *start*) ⇒ void
  start : exact-integer

  Moves backward out of the S-expression containing the position *start*.

24.4 `scheme:text-mixin`

Domain: `text:basic<%>`

Domain: `editor:keymap<%>`

Implements: `text:basic<%>`

Implements: `editor:keymap<%>`

Implements: `scheme:text<%>`

This mixin adds functionality for editing Scheme files.

The result of this mixin uses the same initialization arguments as the mixin's argument.

`after-change-style`

Called after the style is changed for a given range (and after the display is refreshed; use `on-change-style` and `begin-edit-sequence` to avoid extra refreshes when `after-change-style` modifies the editor).

See also `can-change-style?` and `on-edit-sequence`.

No internals locks are set when this method is called.

- **(send** *a-scheme:text-mixin* `after-change-style` *start len***)** ⇒ void
  start : exact non-negative integer
  len : exact non-negative integer

  This calls `end-edit-sequence` and updates the parenthesis hightlight. See `highlight-parens`.

`after-delete`

Called after a given range is deleted from the editor (and after the display is refreshed; use `on-delete` and `begin-edit-sequence` to avoid extra refreshes when `after-delete` modifies the editor).

See also `can-delete?` and `on-edit-sequence`.

No internals locks are set when this method is called.

- **(send** *a-scheme:text-mixin* `after-delete` *start end***)** ⇒ void
  start : exact non-negative integer
  end : exact non-negative integer

  The *start* argument specifies the starting position of the deleted range. The *len* argument specifies number of deleted items (so *start* + *length* is the ending position of the deleted range).

  This calls `end-edit-sequence` and updates the parenthesis hightlight. See `highlight-parens`.

`after-edit-sequence`

Called after a top-level edit sequence completes (involving unnested `begin-edit-sequence` and `end-edit-sequence`).

See also `on-edit-sequence`.

- **(send** *a-scheme:text-mixin* `after-edit-sequence`**)** ⇒ void

  This updates the parenthesis hightlight. See `highlight-parens`.

`after-insert`

Called after items are inserted into the editor (and after the display is refreshed; use `on-insert` and `begin-edit-sequence` to avoid extra refreshes when `after-insert` modifies the editor).

See also `can-insert?` and `on-edit-sequence`.

No internals locks are set when this method is called.

- **(send** *a-scheme:text-mixin* `after-insert` *start len***)** ⇒ void
  start : exact non-negative integer
  len : exact non-negative integer

  The *start* argument specifies the position of the insert. The *len* argument specifies the total length (in positions) of the inserted items.

  This calls `end-edit-sequence` and updates the parenthesis hightlight. See `highlight-parens`.

`after-set-position`

Called after the start and end position have been moved (but not when the position is moved due to inserts or deletes).

See also `on-edit-sequence`.

- **(send** *a-scheme:text-mixin* `after-set-position`**)** ⇒ void
  This calls `end-edit-sequence` and updates the parenthesis hightlight. See `highlight-parens`.

`after-set-size-constraint`

Called after the editor's maximum or minimum height or width is changed (and after the display is refreshed; use `on-set-size-constraint` and `begin-edit-sequence` to avoid extra refreshes when `after-set-size-constraint` modifies the editor).

(This callback method is provided because setting an editor's maximum width may cause lines to be re-flowed with soft carriage returns.)

See also `can-set-size-constraint?` and `on-edit-sequence`.

- **(send** *a-scheme:text-mixin* `after-set-size-constraint`**)** ⇒ void
  This calls `end-edit-sequence` and updates the parenthesis hightlight. See `highlight-parens`.

`get-keymaps`

The keymaps returned from this method are chained to this `editor<%>`'s keymap.

- **(send** *a-scheme:text-mixin* `get-keymaps`**)** ⇒ (listof (instanceof `keymap%`))
  Defaultly returns `(list keymap:get-global)`
  Adds the result of calling `scheme:get-keymap` to the list of keymaps for this object.

`on-close`

This method is called when a frame that shows this buffer is closed.

- **(send** *a-scheme:text-mixin* `on-close`**)** ⇒ void
  Does nothing.
  Removes some preferences callbacks.

`on-focus`

Called when the keyboard focus changes into or out of this editor (and not to/from a snip within the editor) with #t if the focus is being turned on, #f otherwise.

- **(send** *a-scheme:text-mixin* `on-focus` *on?***)** ⇒ boolean
  on? : boolean
  This calls `begin-edit-sequence`.

24.5 scheme:text% = (scheme:text-mixin text:info%)

scheme:text% = (scheme:text-mixin text:info%)

- **(instantiate** scheme:text% () [**(line-spacing** _)] [**(tab-stops** _)] [**(auto-wrap** _)]) ⇒ scheme:text% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

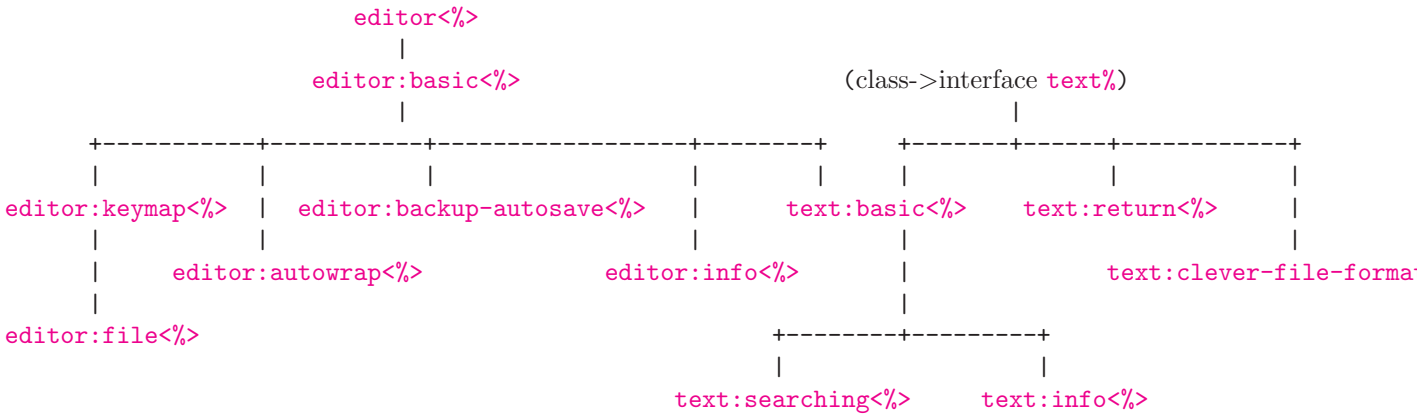
  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

25. Scheme Parenthesis

26. Text

This is the interface hierarchy for the editor and text classes in the framework.



- text:1-pixel-string-snip
- text:1-pixel-tab-snip
- text:autowrap
- text:backup-autosave
- text:basic
- text:clever-file-format
- text:delegate
- text:file
- text:hide-caret/selection
- text:info
- text:keymap
- text:return
- text:searching
- text:basic¡
- text:clever-file-format¡
- text:delegate¡

- text:hide-caret/selection¡
- text:info¡
- text:return¡
- text:searching¡
- text:basic-mixin
- text:clever-file-format-mixin
- text:delegate-mixin
- text:hide-caret/selection-mixin
- text:info-mixin
- text:return-mixin
- text:searching-mixin

26.1 text:basic<%>

Extends: (class->interface text%)

Extends: editor:basic<%>

Classes matching this interface are expected to implement the basic functionality needed by the framework.

get-highlighted-ranges

- (**send** *a-text:basic* get-highlighted-ranges) ⇒ (listof range)
  Returns a list of (opaque) values representing the active ranges in the editor.

get-styles-fixed

If the result of this function is #t, the styles in this text:basic<%> will be fixed. This means that any text inserted to this editor has its style set to this editor's style-list%'s "Standard" style.

See also @milink set-styles-fixed

- (**send** *a-text:basic* get-styles-fixed) ⇒ boolean

highlight-range

This function highlights a region of text in the buffer.

- (**send** *a-text:basic* highlight-range *start end color bitmap caret-space priority*) ⇒ (-¿ void)
  start : exact-integer
  end : exact-integer
  color : (instance color%)

bitmap = #f : (union #f (instance `bitmap%`))
caret-space = #f : boolean
priority = 'low : (union 'high 'low)

The range between *start* and *end* will be highlighted with the color in color, and *bitmap* will be painted over the range of text in black and white. If *bitmap* is `#f`, the range will be inverted, using the platform specific xor. This method is not recommended, because the selection is also displayed using xor.

If *caret-space?* is not `#f`, the left edge of the range will be one pixel short, to leave space for the caret. The caret does not interfere with the right hand side of the range. Note that under X windows the caret is drawn with XOR, which means almost anything can happen. So if the caret is in the middle of the range it may be hard to see, or if it is on the left of the range and *caret-space?* is `#f` it may also be hard to see.

The *priority* argument indicates the relative priority for drawing overlapping regions. If two regions overlap and have different priorities, the region with `'high` priority will be drawn second and only it will be visible in the overlapping region.

This method returns a thunk, which, when invoked, will turn off the highlighting from this range.

`initial-autowrap-bitmap`

The result of this method is used as the initial autowrap bitmap. Override this method to change the initial `bitmap%`. See also `set-autowrap-bitmap`

- (**send** *a-text:basic* `initial-autowrap-bitmap`) ⇒ (union #f (instance `bitmap%`))
  Defaultly returns the result of `icon:get-autowrap-bitmap`

`move/copy-to-edit`

This moves or copies text and snips to another edit.

- (**send** *a-text:basic* `move/copy-to-edit` *dest-text start end dest-pos*) ⇒ void
  dest-text : (instance `text%`)
  start : exact-integer
  end : exact-integer
  dest-pos : exact-integer

  Moves or copies from the edit starting at *start* and ending at *end*. It puts the copied text and snips in *dest-text* starting at location *dest-pos*.

  If a snip refused to be moved, it will be copied, otherwise it will be moved. A snip may refuse to be moved by returning `#f` from `release-from-owner`.

`set-styles-fixed`

Sets the styles fixed parameter of this `text%`. See also `get-styles-fixed`.

- (**send** *a-text:basic* `set-styles-fixed` *fixed?*) ⇒ void
  fixed? : boolean

26.2 `text:basic-mixin`

Domain: `editor:basic<%>`

Domain: (class->interface text%)

Implements: text:basic<%>

Implements: editor:basic<%>

This mixin implements the basic functionality needed for text% objects in the framework.

The class that this mixin produces uses the same initialization arguments as it's input.

- (**instantiate** text:basic-mixin% () [(**line-spacing** _)] [(**tab-stops** _)] [(**auto-wrap** _)]) ⇒ text:basic-mixin% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

after-change-style

Called after the style is changed for a given range (and after the display is refreshed; use on-change-style and begin-edit-sequence to avoid extra refreshes when after-change-style modifies the editor).

See also can-change-style? and on-edit-sequence.

No internals locks are set when this method is called.

- (**send** *a-text:basic-mixin* after-change-style *start len*) ⇒ void
  start : exact non-negative integer
  len : exact non-negative integer

  See set-styles-fixed.

after-insert

Called after items are inserted into the editor (and after the display is refreshed; use on-insert and begin-edit-sequence to avoid extra refreshes when after-insert modifies the editor).

See also can-insert? and on-edit-sequence.

No internals locks are set when this method is called.

- (**send** *a-text:basic-mixin* after-insert *start len*) ⇒ void

start : exact non-negative integer
len : exact non-negative integer

The *start* argument specifies the position of the insert. The *len* argument specifies the total length (in positions) of the inserted items.

See set-styles-fixed.

on-change-style

Called before the style is changed in a given range of the editor, after can-change-style? is called to verify that the change is ok. The after-change-style method is guaranteed to be called after the change has completed.

The editor is internally locked for writing during a call to this method (see also "Locks" (section 8.8, page 169)). Use after-change-style to modify the editor, if necessary.

See also on-edit-sequence.

- **(send** *a-text:basic-mixin* on-change-style *start len*) ⇒ void
  start : exact non-negative integer
  len : exact non-negative integer

  See set-styles-fixed.

on-insert

Called before items are inserted into the editor, after can-insert? is called to verify that the insertion is ok. The after-insert method is guaranteed to be called after the insert has completed.

The editor is internally locked for writing during a call to this method (see also "Locks" (section 8.8, page 169)). Use after-insert to modify the editor, if necessary.

See also on-edit-sequence.

- **(send** *a-text:basic-mixin* on-insert *start end*) ⇒ void
  start : exact-int
  end : exact-int

  The *start* argument specifies the position of the insert. The *len* argument specifies the total length (in positions) of the items to be inserted.

  See set-styles-fixed.

on-paint

Provides a way to add arbitrary graphics to an editor's display. This method is called just before and just after every painting of the editor.

The on-paint method, together with the snips' draw methods, must be able to draw the entire state of an editor. Never paint directly into an editor's display canvas except from within on-paint or draw. Instead, put all extra drawing code within on-paint and call invalidate-bitmap-cache when part of the display needs to be repainted.

If an on-paint method uses cached location information, then the cached information should be recomputed in response to a call of invalidate-bitmap-cache.

The on-paint method must not make any assumptions about the state of the drawing context (e.g., the current pen), except that the clipping region is already set to something appropriate. Before on-paint returns, it must restore any drawing context settings that it changes.

The editor is internally locked for writing and reflowing during a call to this method (see also "Locks" (section 8.8, page 169)).

- **(send** *a-text:basic-mixin* on-paint *before? dc left top right bottom dx dy draw-caret*) ⇒ void
  before? : boolean
  dc : dc<%> object
  left : real number
  top : real number
  right : real number
  bottom : real number
  dx : real number
  dy : real number
  draw-caret : symbol in '(no-caret show-inactive-caret show-caret)

  The *before?* argument is #t when the method is called just before a painting the contents of the editor or #f when it is called after painting. The *left*, *top*, *right*, and *bottom* arguments specify which region of the editor is being repainted, in editor coordinates. To get the coordinates for *dc*, offset editor coordinates by adding (*dx*, *dy*). See "Caret" (section 8.5, page 168) for information about *draw-caret*.

  See also invalidate-bitmap-cache.

  Draws the rectangles installed by highlight-range.

26.3 text:hide-caret/selection<%>

Extends: text:basic<%>

This class hides the caret, except when the selection is active.

Instances of this class are useful for editors that used for displaying purposes, but still allow users to copy their text.

26.4 text:hide-caret/selection-mixin

Domain: text:basic<%>

Implements: text:basic<%>

Implements: text:hide-caret/selection<%>

after-set-position

Called after the start and end position have been moved (but not when the position is moved due to inserts or deletes).

See also on-edit-sequence.

- (**send** *a-text:hide-caret/selection-mixin* `after-set-position`) ⇒ void

  Calls `hide-caret` to hide the caret when there is only a caret and no selection.

26.5 `text:searching<%>`

Extends: `text:basic<%>`

Extends: `editor:keymap<%>`

Any object matching this interface can be searched.

26.6 `text:searching-mixin`

Domain: `text:basic<%>`

Domain: `editor:keymap<%>`

Implements: `text:basic<%>`

Implements: `editor:keymap<%>`

Implements: `text:searching<%>`

This `text%` can be searched.

The result of this mixin uses the same initialization arguments as the mixin's argument.

`get-keymaps`

The keymaps returned from this method are chained to this `editor<%>`'s keymap.

- (**send** *a-text:searching-mixin* `get-keymaps`) ⇒ (list-of (instance `keymap%`))

  Defaultly returns `(list keymap:get-global)`

  This returns a list containing the super-class's keymaps, plus the result of `keymap:get-search`

26.7 `text:return<%>`

Extends: `(class->interface text%)`

Objects supporting this interface were created by `text:return-mixin`.

26.8 text:return-mixin

Domain: (class->interface text%)

Implements: text:return<%>

Use this buffer to perform some special action when return is typed.

- (make-object text:return-mixin% *return*) ⇒ text:return-mixin% object
  return : (-¿ boolean)

on-local-char

Called by on-char when the event is *not* handled by a caret-owning snip.

Consider overriding on-default-char instead of this method.

- (**send** *a-text:return-mixin* on-local-char *event*) ⇒ void
  event : key-event% object

  Either lets the keymap handle the event or calls on-default-char .

  If *key* is either return or newline, only invoke the *return* thunk (initialization argument) and do nothing else.

26.9 text:delegate<%>

Extends: text:basic<%>

Implementations of this interface copy all of the changes to this editor to the result of get-delegate except instead of regular string and tab snips, instead instances of text:1-pixel-string-snip% and text:1-pixel-tab-snip% are created.

The contents of the two editor are kept in sync, as modifications to this object happen.

get-delegate

The result of this method is the text% object that the contents of this editor are being delegated to, or #f, if there is none.

- (**send** *a-text:delegate* get-delegate) ⇒ (union #f (instanceof text%))

set-delegate

This method sets the current delegate.

- (**send** *a-text:delegate* set-delegate *delegate*) ⇒ void
  delegate : (union #f (instanceof text%))

When it is set, all of the snips are copied from this object to *delegate*. Additionally, if this object implements scheme:text<%> the tab settings of *delegate* are updated to match this objects.

26.10 text:1-pixel-string-snip%

Superclass: string-snip%

This class re-uses the implementation of string-snip% to implement a string snip that just draws a single pixel for each character in the string.

See also text:1-pixel-tab-snip% for a similar extension to the tab-snip%class.

This snip is used in conjunction with the frame:delegate<%> and text:delegate<%> interfaces.

- (make-object text:1-pixel-string-snip% *allocsize*) ⇒ text:1-pixel-string-snip% object
  allocsize = 0 : exact non-negative integer

  Creates an empty string snip. The *allocsize* argument is a hint about how much storage space for text should be initially allocated by the snip.

- (make-object text:1-pixel-string-snip% *s*) ⇒ text:1-pixel-string-snip% object
  s : string

  Creates a string snip with the given initial string.

copy

Creates and returns a copy of this snip. The copy method is responsible for copying this snip's style (as returned by get-style) to the new snip.

- (**send** *a-text:1-pixel-string-snip* copy) ⇒ (instanceof snip%)

  Creates and returns an instance of text:1-pixel-string-snip%.

draw

Called (by an editor) to draw the snip.

Before this method is called, the correct font, text color, and pen color will have been set in the drawing context for this snip already. (This is *not* true for get-extent or partial-offset.) The draw method must not make any other assumptions about the state of the drawing context, except that the clipping region is already set to something appropriate. Before draw returns, it must restore any drawing context settings that it changes.

See also on-paint in editor<%>.

The snip's editor is usually internally locked for writing and reflowing when this method is called (see also "Locks" (section 8.8, page 169)).

- (**send** *a-text:1-pixel-string-snip* draw *dc x y left top right bottom dx dy draw-caret*) ⇒ void
  dc : (instanceof dc<%>)
  x : real number
  y : real number
  left : real number

top : real number
right : real number
bottom : real number
dx : real number
dy : real number
draw-caret : (union 'no-caret 'show-inactive-caret 'show-caret)

Draws black pixels for non-whitespace characters and draws nothing for whitespace characters.

`get-extent`

Calculates the snip's width, height, descent (amount of height which is drawn below the baseline), space (amount of height which is "filler" space at the top), and horizontal spaces (amount of width which is "filler" space at the left and right).

This method is called by the snip's administrator; it should not be called directly by others. To get the extent of a snip, use `get-snip-location` in `editor<%>` .

A drawing context is provided for the purpose of finding font sizes, but no drawing should occur. The `get-extent` and `partial-offset` methods must not make any assumptions about the state of the drawing context, except that it is scaled properly. In particular, the font for the snip's style is not automatically set in the drawing context before the method is called.[1] If `get-extent` or `partial-offset` changes the drawing context's setting, it must restore them before returning. However, the methods should not need to change the drawing context; only font settings can affect measurement results from a device context, and `get-text-extent` in `dc<%>` accepts a `font%` argument for sizing that overrides that device context's current font.

The snip's left and top locations are provided in editor coordinates. In a text editor, the y-coordinate is the *line's* top location; the snip's actual top location is potentially undetermined until its height is known.

If a snip caches the result size for future replies, it should invalidate its cached size when `size-cache-invalid` is called (especially if the snip's size depends on any device context properties).

If a snip's size changes after receiving a call to `get-extent` and before receiving a call to `size-cache-invalid`, then the snip must notify its administrator of the size change, so that the administrator can recompute its derived size information. Notify the administrator of a size change by call its `resized` method.

The snip's editor is usually internally locked for writing and reflowing when this method is called (see also "Locks" (section 8.8, page 169)).

- (**send** *a-text:1-pixel-string-snip* `get-extent` *dc x y w h descent space lspace rspace*) ⇒ void
  dc : (instanceof `dc<%>`)
  x : real number
  y : real number
  w = #f : (box (union non-negative-real-number #f))
  h = #f : (box (union non-negative-real-number #f))
  descent = #f : (box (union non-negative-real-number #f))
  space = #f : (box (union non-negative-real-number #f))
  lspace = #f : (box (union non-negative-real-number #f))
  rspace = #f : (box (union non-negative-real-number #f))

  Sets the descent, space, lspace, and rspace to zero. Sets the height to 1. Sets the width to the number of characters in the string.

[1] Many snips cache their size information, so automatically setting the font would be wasteful.

`insert`

Inserts text into the snip. The system can insert text into a text snip without calling this method.

- (**send** *a-text:1-pixel-string-snip* `insert` *s len pos*) ⇒ void
  s : string
  len : exact non-negative integer
  pos = 0 : exact non-negative integer

  Inserts *s* (with length *len*) into the snip at relative position *pos* within the snip.

`split`

Splits the snip into two snips. This is called when a snip has more than one item and something is inserted between two items.

The arguments are a relative position integer and two boxes. The position integer specifies how many items should be given to the new first snip; the rest go to the new second snip. The two boxes must be filled with two new snips. (The old snip is no longer used, so it can be recycled as a new snip.)

If the returned snips do not have the expected counts, their counts are forcibly modified. If either returned snip is already owned by a another administrator, a surrogate snip is created.

The snip's editor is usually internally locked for reading when this method is called (see also "Locks" (section 8.8, page 169)).

- (**send** *a-text:1-pixel-string-snip* `split` *position first second*) ⇒ void
  position : exact non-negative integer
  first : (box (instanceof `snip%`))
  second : (box (instanceof `snip%`))

  Fills the boxes with instance of `text:1-pixel-string-snip%`s.

26.11 `text:1-pixel-tab-snip%`

Superclass: `tab-snip%`

This class re-uses the implementation of `tab-snip%` to implement a string snip that is always one pixel high.

See also `text:1-pixel-string-snip%` for a similar extension to the `string-snip%`class.

This snip is used in conjunction with the `frame:delegate<%>` and `text:delegate<%>` interfaces.

- (`make-object text:1-pixel-tab-snip%`) ⇒ `text:1-pixel-tab-snip%` object

  Creates a snip for a single tab, though the tab is initially empty.

  Normally, a single tab is inserted into a `tab-snip%` object using the `insert` method.

  The tab's content is not drawn, through it is used when determining the size of a single character in editors where tabbing is determined by the character width (see `set-tabs`); if the content is a single tab character (the normal case), then the average character width of snip's font is used as the tab's width.

`copy`

Creates and returns a copy of this snip. The `copy` method is responsible for copying this snip's style (as returned by `get-style`) to the new snip.

- (**send** *a-text:1-pixel-tab-snip* `copy`) ⇒ (instanceof `snip%`)
  Creates and returns an instance of `text:1-pixel-tab-snip%`.

`draw`

Called (by an editor) to draw the snip.

Before this method is called, the correct font, text color, and pen color will have been set in the drawing context for this snip already. (This is *not* true for `get-extent` or `partial-offset`.) The `draw` method must not make any other assumptions about the state of the drawing context, except that the clipping region is already set to something appropriate. Before `draw` returns, it must restore any drawing context settings that it changes.

See also `on-paint` in `editor<%>`.

The snip's editor is usually internally locked for writing and reflowing when this method is called (see also "Locks" (section 8.8, page 169)).

- (**send** *a-text:1-pixel-tab-snip* `draw` *dc x y left top right bottom dx dy draw-caret*) ⇒ void
  dc : (instanceof `dc<%>`)
  x : real number
  y : real number
  left : real number
  top : real number
  right : real number
  bottom : real number
  dx : real number
  dy : real number
  draw-caret : (union 'no-caret 'show-inactive-caret 'show-caret)
  Draws nothing.

`get-extent`

Calculates the snip's width, height, descent (amount of height which is drawn below the baseline), space (amount of height which is "filler" space at the top), and horizontal spaces (amount of width which is "filler" space at the left and right).

This method is called by the snip's administrator; it should not be called directly by others. To get the extent of a snip, use `get-snip-location` in `editor<%>` .

A drawing context is provided for the purpose of finding font sizes, but no drawing should occur. The `get-extent` and `partial-offset` methods must not make any assumptions about the state of the drawing context, except that it is scaled properly. In particular, the font for the snip's style is not automatically set in the drawing context before the method is called.[2] If `get-extent` or `partial-offset` changes the drawing context's setting, it must restore them before returning. However, the methods should not need to change the drawing context; only font settings can affect measurement results from a device context, and

[2]Many snips cache their size information, so automatically setting the font would be wasteful.

get-text-extent in dc<%> accepts a font% argument for sizing that overrides that device context's current font.

The snip's left and top locations are provided in editor coordinates. In a text editor, the y-coordinate is the *line's* top location; the snip's actual top location is potentially undetermined until its height is known.

If a snip caches the result size for future replies, it should invalidate its cached size when size-cache-invalid is called (especially if the snip's size depends on any device context properties).

If a snip's size changes after receiving a call to get-extent and before receiving a call to size-cache-invalid, then the snip must notify its administrator of the size change, so that the administrator can recompute its derived size information. Notify the administrator of a size change by call its resized method.

The snip's editor is usually internally locked for writing and reflowing when this method is called (see also "Locks" (section 8.8, page 169)).

- (**send** *a-text:1-pixel-tab-snip* get-extent *dc x y w h descent space lspace rspace*) ⇒ void
  dc : (instanceof dc<%>)
  x : real number
  y : real number
  w = #f : (box (union non-negative-real-number #f))
  h = #f : (box (union non-negative-real-number #f))
  descent = #f : (box (union non-negative-real-number #f))
  space = #f : (box (union non-negative-real-number #f))
  lspace = #f : (box (union non-negative-real-number #f))
  rspace = #f : (box (union non-negative-real-number #f))

  Sets the descent, space, lspace, and rspace to zero. Sets the height to 1. Sets the width to the width of tabs as returned in the tab-width parameter of the get-tabs method.

split

Splits the snip into two snips. This is called when a snip has more than one item and something is inserted between two items.

The arguments are a relative position integer and two boxes. The position integer specifies how many items should be given to the new first snip; the rest go to the new second snip. The two boxes must be filled with two new snips. (The old snip is no longer used, so it can be recycled as a new snip.)

If the returned snips do not have the expected counts, their counts are forcibly modified. If either returned snip is already owned by a another administrator, a surrogate snip is created.

The snip's editor is usually internally locked for reading when this method is called (see also "Locks" (section 8.8, page 169)).

- (**send** *a-text:1-pixel-tab-snip* split *position first second*) ⇒ void
  position : exact non-negative integer
  first : (box (instanceof snip%))
  second : (box (instanceof snip%))

  Fills the boxes with instance of text:1-pixel-tab-snip%s.

26.12 text:delegate-mixin

Domain: text:basic<%>

Implements: text:basic<%>

Implements: text:delegate<%>

This mixin provides an implementation of the text:delegate<%> interface.

after-change-style

Called after the style is changed for a given range (and after the display is refreshed; use on-change-style and begin-edit-sequence to avoid extra refreshes when after-change-style modifies the editor).

See also can-change-style? and on-edit-sequence.

No internals locks are set when this method is called.

- (**send** *a-text:delegate-mixin* after-change-style *start len*) ⇒ void
  start : number
  len : number
  forwards the changed style to the delegate.

after-delete

Called after a given range is deleted from the editor (and after the display is refreshed; use on-delete and begin-edit-sequence to avoid extra refreshes when after-delete modifies the editor).

See also can-delete? and on-edit-sequence.

No internals locks are set when this method is called.

- (**send** *a-text:delegate-mixin* after-delete *start len*) ⇒ void
  start : number
  len : number
  The *start* argument specifies the starting position of the deleted range. The *len* argument specifies number of deleted items (so *start* + *length* is the ending position of the deleted range).
  forwards the change to the delegate.

after-edit-sequence

Called after a top-level edit sequence completes (involving unnested begin-edit-sequence and end-edit-sequence).

See also on-edit-sequence.

- (**send** *a-text:delegate-mixin* after-edit-sequence) ⇒ void
  ends an edit sequence in the delegate.

`after-insert`

Called after items are inserted into the editor (and after the display is refreshed; use `on-insert` and `begin-edit-sequence` to avoid extra refreshes when `after-insert` modifies the editor).

See also `can-insert?` and `on-edit-sequence`.

No internals locks are set when this method is called.

- (**send** *a-text:delegate-mixin* `after-insert` *start len*) ⇒ void
  start : number
  len : number

  The *start* argument specifies the position of the insert. The *len* argument specifies the total length (in positions) of the inserted items.

  forwards the change to the delegate

`after-load-file`

Called just after the editor is loaded from a file.

The argument to the method originally specified whether the save was successful, but failures now trigger exceptions such that the method is not even called. Consequently, the argument is always #t.

See also `can-load-file?` and `on-load-file`.

- (**send** *a-text:delegate-mixin* `after-load-file` *sucess?*) ⇒ void
  sucess? : boolean

  updates the delegate with the new contents of the text.

`highlight-range`

This function highlights a region of text in the buffer.

- (**send** *a-text:delegate-mixin* `highlight-range` *start end color bitmap caret-space priority*) ⇒ (-¿ void)
  start : exact-integer
  end : exact-integer
  color : (instance `color%`)
  bitmap : (union #f (instance `bitmap%`))
  caret-space = #f : boolean
  priority = 'low : (union 'high 'low)

  The range between *start* and *end* will be highlighted with the color in color, and *bitmap* will be painted over the range of text in black and white. If *bitmap* is `#f`, the range will be inverted, using the platform specific xor. This method is not recommended, because the selection is also displayed using xor.

  If *caret-space?* is not `#f`, the left edge of the range will be one pixel short, to leave space for the caret. The caret does not interfere with the right hand side of the range. Note that under X windows the caret is drawn with XOR, which means almost anything can happen. So if the caret is in the middle of the range it may be hard to see, or if it is on the left of the range and *caret-space?* is `#f` it may also be hard to see.

The *priority* argument indicates the relative priority for drawing overlapping regions. If two regions overlap and have different priorities, the region with `'high` priority will be drawn second and only it will be visible in the overlapping region.

This method returns a thunk, which, when invoked, will turn off the highlighting from this range.

In addition to calling the super method, `highlight-range`, this method forwards the highlighting to the delegatee.

`on-edit-sequence`

Called just before a top-level (i.e., unnested) edit sequence starts.

During an edit sequence, all callbacks methods are invoked normally, but it may be appropriate for these callbacks to delay computation during an edit sequence. The callbacks must manage this delay manually. Thus, when overriding other callback methods, such as `on-insert` in `text%`, `on-insert` in `pasteboard%`, `after-insert` in `text%`, or `after-insert` in `pasteboard%`, consider overriding `on-edit-sequence` and `after-edit-sequence` as well.

"Top-level edit sequence" refers to an outermost pair of `begin-edit-sequence` and `end-edit-sequence` calls. The embedding of an editor within another editor does not affect the timing of calls to `on-edit-sequence`, even if the embedding editor is in an edit sequence.

- **(send** *a-text:delegate-mixin* `on-edit-sequence`**)** ⇒ void

  starts an edit sequence in the delegate.

`on-load-file`

Called just before the editor is loaded from a file, after calling `can-load-file?` to verify that the load is allowed. See also `after-load-file`.

- **(send** *a-text:delegate-mixin* `on-load-file` *filename format***)** ⇒ void
  filename : string
  format : symbol

  The *filename* argument is the name the file will be loaded from. See `load-file` for information about *format*.

  remembers the filename, for use in `after-load-file`.

`on-paint`

Provides a way to add arbitrary graphics to an editor's display. This method is called just before and just after every painting of the editor.

The `on-paint` method, together with the snips' `draw` methods, must be able to draw the entire state of an editor. Never paint directly into an editor's display canvas except from within `on-paint` or `draw`. Instead, put all extra drawing code within `on-paint` and call `invalidate-bitmap-cache` when part of the display needs to be repainted.

If an `on-paint` method uses cached location information, then the cached information should be recomputed in response to a call of `invalidate-bitmap-cache`.

The `on-paint` method must not make any assumptions about the state of the drawing context (e.g., the

current pen), except that the clipping region is already set to something appropriate. Before `on-paint` returns, it must restore any drawing context settings that it changes.

The editor is internally locked for writing and reflowing during a call to this method (see also "Locks" (section 8.8, page 169)).

- (**send** *a-text:delegate-mixin* `on-paint` *before? dc left top right bottom dx dy draw-caret*) ⇒ void
  before? : boolean
  dc : `dc<%>` object
  left : real number
  top : real number
  right : real number
  bottom : real number
  dx : real number
  dy : real number
  draw-caret : symbol in `'(no-caret show-inactive-caret show-caret)`

  The *before?* argument is #t when the method is called just before a painting the contents of the editor or #f when it is called after painting. The *left*, *top*, *right*, and *bottom* arguments specify which region of the editor is being repainted, in editor coordinates. To get the coordinates for *dc*, offset editor coordinates by adding (*dx*, *dy*). See "Caret" (section 8.5, page 168) for information about *draw-caret*.

  See also `invalidate-bitmap-cache`.

  Draws a blue region in the delegatee editor that shows where the visible region of the delegate editor is.

26.13 `text:info<%>`

Extends: `text:basic<%>`

Objects supporting this interface are expected to send information about themselves to the frame that is displaying them.

26.14 `text:info-mixin`

Domain: `text:basic<%>`

Domain: `editor:keymap<%>`

Implements: `text:basic<%>`

Implements: `editor:keymap<%>`

Implements: `text:info<%>`

This mixin adds support for supplying information to objects created with `frame:info-mixin`. When this `editor:basic<%>` is displayed in a frame, that frame must have been created with `frame:info-mixin`.

`after-delete`

Called after a given range is deleted from the editor (and after the display is refreshed; use `on-delete` and `begin-edit-sequence` to avoid extra refreshes when `after-delete` modifies the editor).

See also `can-delete?` and `on-edit-sequence`.

No internals locks are set when this method is called.

- **(send** *a-text:info-mixin* `after-delete` *start end***)** ⇒ void
  start : exact non-negative integer
  end : exact non-negative integer

  The *start* argument specifies the starting position of the deleted range. The *len* argument specifies number of deleted items (so *start* + *length* is the ending position of the deleted range).

  Calls the `editor-position-changed` method of the frame that is viewing this object. It uses `get-canvas` to get the canvas for this frame, and uses that canvas's `top-level-window<%>` as the frame.

`after-insert`

Called after items are inserted into the editor (and after the display is refreshed; use `on-insert` and `begin-edit-sequence` to avoid extra refreshes when `after-insert` modifies the editor).

See also `can-insert?` and `on-edit-sequence`.

No internals locks are set when this method is called.

- **(send** *a-text:info-mixin* `after-insert` *start len***)** ⇒ void
  start : exact non-negative integer
  len : exact non-negative integer

  The *start* argument specifies the position of the insert. The *len* argument specifies the total length (in positions) of the inserted items.

  Calls the `editor-position-changed` method of the frame that is viewing this object. It uses `get-canvas` to get the canvas for this frame, and uses that canvas's `top-level-window<%>` as the frame.

`after-set-position`

Called after the start and end position have been moved (but not when the position is moved due to inserts or deletes).

See also `on-edit-sequence`.

- **(send** *a-text:info-mixin* `after-set-position`**)** ⇒ void

  Calls the `editor-position-changed` method of the frame that is viewing this object. It uses `get-canvas` to get the canvas for this frame, and uses that canvas's `top-level-window<%>` as the frame.

`set-anchor`

Turns anchoring on or off. This method can be overridden to affect or detect changes in the anchor state. See also `get-anchor`.

- **(send** *a-text:info-mixin* `set-anchor` *on?***)** ⇒ void
  on? : boolean

  If *on?* is not #f, then the selection will be automatically extended when cursor keys are used (or, more generally, when `move-position` is used to move the selection), otherwise anchoring is turned off. Anchoring is automatically turned off if the user does anything besides cursor movements.

  Calls the `anchor-status-changed` method of the frame that is viewing this object. It uses `get-canvas` to get the canvas for this frame, and uses that canvas's `top-level-window<%>` as the frame.

`set-overwrite-mode`

Enables or disables overwrite mode. See `get-overwrite-mode`. This method can be overridden to affect or detect changes in the overwrite mode.

- **(send** *a-text:info-mixin* `set-overwrite-mode` *on?***)** ⇒ void
  on? : boolean

  Calls the `overwrite-status-changed`method of the frame that is viewing this object. It uses `get-canvas` to get the canvas for this frame, and uses that's `top-level-window<%>` as the frame.

26.15 `text:clever-file-format<%>`

Extends: `(class->interface text%)`

Objects supporting this interface are expected to support a clever file format when saving.

26.16 `text:clever-file-format-mixin`

Domain: `(class->interface text%)`

Implements: `text:clever-file-format<%>`

The result of this mixin uses the same initialization arguments as the mixin's argument.

When files are saved from this `text%`, a check is made to see if there are any non-`string-snip%` objects in the `text%`. If so, it is saved using the file format `'std`. (see `set-file-format` for more information. If not, the file format passed to `save-file` is used.

- **(instantiate** `text:clever-file-format-mixin%` **()** `[(line-spacing _)]` `[(tab-stops _)]` `[(auto-wrap _)])` ⇒ `text:clever-file-format-mixin%` object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

See set-tabs for information about *tabstops*.

If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

`on-save-file`

Called just before the editor is saved to a file, after calling can-save-file? to verify that the save is allowed. See also after-save-file.

- **(send** *a-text:clever-file-format-mixin* `on-save-file` *filename format*) ⇒ void
  filename : string
  format : symbol in `'(guess standard text text-force-cr same copy)`

  The *filename* argument is the name the file will be saved to. See load-file for information about *format*.

  If the method get-file-format returns `'standard` and the text has only string-snip%s, the file format is set to `'text`.

  If the method get-file-format returns `'text` and the text has some non string-snip%s, the file format is set to `'standard`.

  Depending on the user's preferences, the user may also be queried.

  Also, the chagnes to the file format only happen if the argument *file-format* is `'copy` or `'same`.

26.17 text:basic% = (editor:basic-mixin (text:basic-mixin text%))

`text:basic% = (editor:basic-mixin (text:basic-mixin text%))`

- **(instantiate** `text:basic% () [(line-spacing _)] [(tab-stops _)] [(auto-wrap _)]`) ⇒ text:basic% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

26.18 text:hide-caret/selection% = (text:hide-caret/selection-mixin text:basic%)

text:hide-caret/selection% = (text:hide-caret/selection-mixin text:basic%)

- (**instantiate** text:hide-caret/selection% () [(**line-spacing** _)] [(**tab-stops** _)] [(**auto-wrap** _)]) ⇒ text:hide-caret/selection% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

26.19 text:delegate% = (text:delegate-mixin text:basic%)

text:delegate% = (text:delegate-mixin text:basic%)

- (**instantiate** text:delegate% () [(**line-spacing** _)] [(**tab-stops** _)] [(**auto-wrap** _)]) ⇒ text:delegate% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

26.20 text:keymap% = (editor:keymap-mixin text:basic%)

text:keymap% = (editor:keymap-mixin text:basic%)

- (**instantiate** text:keymap% () [(**line-spacing** _)] [(**tab-stops** _)] [(**auto-wrap** _)]) ⇒ text:keymap% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

See set-tabs for information about *tabstops*.

If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

26.21 text:return% = (text:return-mixin text:keymap%)

text:return% = (text:return-mixin text:keymap%)

- (**instantiate** text:return% () [(**line-spacing** _)] [(**tab-stops** _)] [(**auto-wrap** _)]) ⇒ text:return% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

- (make-object text:return% *return*) ⇒ text:return% object
  return : (-¿ boolean)

26.22 text:autowrap% = (editor:autowrap-mixin text:keymap%)

text:autowrap% = (editor:autowrap-mixin text:keymap%)

- (**instantiate** text:autowrap% () [(**line-spacing** _)] [(**tab-stops** _)] [(**auto-wrap** _)]) ⇒ text:autowrap% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

26.23 `text:file% = (editor:file-mixin text:autowrap%)`

`text:file% = (editor:file-mixin text:autowrap%)`

- **(instantiate** `text:file%` `()` `[(line-spacing _)]` `[(tab-stops _)]` `[(auto-wrap _)]`**)** ⇒ `text:file%` object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See `set-tabs` for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via `auto-wrap` .

  A new `keymap%` object is created for the new editor. See also `get-keymap` and `set-keymap`.

  A new `style-list%` object is created for the new editor. See also `get-style-list` and `set-style-list`.

26.24 `text:clever-file-format% = (text:clever-file-format-mixin text:file%)`

`text:clever-file-format% = (text:clever-file-format-mixin text:file%)`

- **(instantiate** `text:clever-file-format%` `()` `[(line-spacing _)]` `[(tab-stops _)]` `[(auto-wrap _)]`**)** ⇒ `text:clever-file-format%` object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See `set-tabs` for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via `auto-wrap` .

  A new `keymap%` object is created for the new editor. See also `get-keymap` and `set-keymap`.

  A new `style-list%` object is created for the new editor. See also `get-style-list` and `set-style-list`.

26.25 `text:backup-autosave% = (editor:backup-autosave-mixin text:clever-file-format%)`

`text:backup-autosave% = (editor:backup-autosave-mixin text:clever-file-format%)`

- **(instantiate** `text:backup-autosave%` `()` `[(line-spacing _)]` `[(tab-stops _)]` `[(auto-wrap _)]`**)** ⇒ `text:backup-autosave%` object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

See set-tabs for information about *tabstops*.

If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

26.26 text:searching% = (text:searching-mixin text:backup-autosave%)

```
text:searching% = (text:searching-mixin text:backup-autosave%)
```

- **(instantiate** text:searching% **() [(line-spacing _)] [(tab-stops _)] [(auto-wrap _)])** ⇒ text:searching% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

26.27 text:info% = (editor:info-mixin (text:info-mixin text:searching%))

```
text:info% = (editor:info-mixin (text:info-mixin text:searching%))
```

- **(instantiate** text:info% **() [(line-spacing _)] [(tab-stops _)] [(auto-wrap _)])** ⇒ text:info% object
  line-spacing = 1.0 : non-negative real number
  tab-stops = null : list of real numbers
  auto-wrap = #f : boolean

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See set-tabs for information about *tabstops*.

  If *auto-wrap* is true, then auto-wrapping is enabled via auto-wrap .

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

27. Version

28. Framework Functions

28.1 Framework Functions

`application:current-app-name`

- (`application:current-app-name`) ⇒ string?
- (`application:current-app-name` *name*) ⇒ void?
  name : string?

  This is a parameter specifying the name of the current application. It is used in the help menu (see `frame:standard-menus%`) and in frame titles (see `frame:editor%`).

  The first case in the case-lambda returns the current name, and the second case in the case-lambda sets the name of the application to *name*.

`autosave:register`

- (`autosave:register` *obj*) ⇒ void?
  obj : (is-a?/c `autosave:autosavable<%>`)

  Adds *obj* to the list of objects to be autosaved. When it is time to autosave, the `do-autosave` method of the object is called. This method is responsible for performing the autosave.

  There is no need to de-register an object because the autosaver keeps a "weak" pointer to the object; i.e., the autosaver does not keep an object from garbage collection.

`color-model:rgb->xyz`

- (`color-model:rgb->xyz` *r* *g* *b*) ⇒ color-model:xyz?
  r : number?
  g : number?
  b : number?

  Converts a color represented as a red-green-blue tuple (each value from 0 to 255) into an XYZ tuple. This describes a point in the CIE XYZ color space.

`color-model:rgb-color-distance`

- (`color-model:rgb-color-distance` *red-a* *green-a* *blue-a* *red-b* *green-b* *blue-b*) ⇒ number?
  red-a : number?
  green-a : number?
  blue-a : number?
  red-b : number?
  green-b : number?
  blue-b : number?

This calculates a distance between two colors. The smaller the distance, the closer the colors should appear to the human eye. A distance of 10 is reasonably close that it could be called the same color.

This function is not symmetric in red, green, and blue, so it is important to pass red, green, and blue components of the colors in the the proper order. The first three arguments are red, green and blue for the first color, respectively, and the second three arguments are red green and blue for the second color, respectively.

`color-model:xyz->rgb`

- (`color-model:xyz->rgb` *x y z*) ⇒ (list/p number? number? number?)
  x : number?
  y : number?
  z : number?

  Converts an XYZ-tuple (in the CIE XYZ colorspace) into a list of values representing an RGB-tuple.

`color-model:xyz-x`

- (`color-model:xyz-x` *xyz*) ⇒ number?
  xyz : color-model:xyz?

  Extracts the x component of *xyz*.

`color-model:xyz-y`

- (`color-model:xyz-y` *xyz*) ⇒ number?
  xyz : color-model:xyz?

  Extracts the y component of *xyz*.

`color-model:xyz-z`

- (`color-model:xyz-z` *xyz*) ⇒ number?
  xyz : color-model:xyz?

  Extracts the z component of *xyz*.

`color-model:xyz?`

- (`color-model:xyz?` *val*) ⇒ boolean?
  val : any?

  Determines if *val* an xyz color record.

`exit:can-exit?`

- (`exit:can-exit?` *skip-user-query?*) ⇒ void?
  skip-user-query? : boolean?

  Calls the "can-callbacks". See `exit:insert-can?-callback` for more information.

  If *skip-user-query?* is `#f`, and the preference `'framework:verify-exit` is not `#f`, (see section 23 for more info about preferences) this procedure asks the user if they want to exit. Otherwise it doesn't ask the user.

exit:exit

- (exit:exit *skip-user-query?*) ⇒ any?
  skip-user-query? = #f : boolean?

  exit:exit performs three actions:

  – invokes the exit-callbacks, with exit:can-exit?If none of the "can?" callbacks return #f,
  – invokes exit:on-exitand then
  – exit (a mzscheme procedure).

  Passes *skip-user-query?* to exit:can-exit?.

exit:frame-exiting

- (exit:frame-exiting *frame*) ⇒ void?
  frame : (union false? (is-a?/c frame%) (is-a?/c dialog%))

- (exit:frame-exiting) ⇒ (union false? (is-a?/c frame%) (is-a?/c dialog%))

  This is a parameter whose value is used as the parent of the "Are you sure you want to exit" dialog.

  The first case of the case-lambda sets the value of the parameter to *frame*. The second case of the case-lambda returns the current value of the parameter.

exit:insert-can?-callback

- (exit:insert-can?-callback *callback*) ⇒ (-> void?)
  callback : (-> boolean?)

  Use this function to add a callback that determines if an attempted exit can proceed. This callback should not clean up any state, since another callback may veto the exit. Use exit:insert-on-callback for callbacks that clean up state.

exit:insert-on-callback

- (exit:insert-on-callback *callback*) ⇒ (-> void?)
  callback : (-> void?)

  Adds a callback to be called when exiting. This callback must not fail. If a callback should stop an exit from happening, use exit:insert-can?-callback.

exit:on-exit

- (exit:on-exit) ⇒ void?

  Calls the "on-callbacks". See exit:insert-on-callback for more information.

exn:exn?

- (exn:exn? *exn*) ⇒ boolean?
  exn : any?

  Tests if a value is a framework exception.

`exn:make-exn`

- `(exn:make-exn` *message continuation-marks*`)` ⇒ exn?
  message : string?
  continuation-marks : continuation-mark-set?

  Creates a framework exception.

`exn:make-unknown-preference`

- `(exn:make-unknown-preference` *message continuation-marks*`)` ⇒ exn:unknown-preference?
  message : string?
  continuation-marks : continuation-mark-set?

  Creates an unknown preference exception.

`exn:unknown-preference?`

- `(exn:unknown-preference?` *exn*`)` ⇒ boolean?
  exn : any?

  Determines if a value is an unknown preference exn.

`finder:common-get-file`

- `(finder:common-get-file` *directory prompt filter filter-msg parent*`)` ⇒ (union string? false?)
  directory = #f : (union string? false?)
  prompt = "Select File" : string?
  filter = #f : (union regexp? false?)
  filter-msg = "That filename does not have the right form." : string?
  parent = #f : (union false? (is-a?/c `top-level-window<%>`))

  This procedure queries the user for a single filename, using a platform-independent dialog box. Consider using `finder:get-file` instead of this function.

  See section 10 for more information.

`finder:common-get-file-list`

- `(finder:common-get-file-list` *directory prompt filter filter-msg parent*`)` ⇒ (union string? false?)
  directory = #f : (union false? string?)
  prompt = "Select File" : string?
  filter = #f : (union false? regexp?)
  filter-msg = "That filename does not have the right form." : string?
  parent = #f : (union false? (is-a?/c `top-level-window<%>`))

  This procedure queries the user for a list of filenames, using a platform-independent dialog box.

  See section 10 for more information.

`finder:common-put-file`

- `(finder:common-put-file` *name directory replace? prompt filter filter-msg parent*`)` ⇒ (union false? string?)
  name = "Untitled" : string?
  directory = #f : (union false? string?)

replace? = #f : boolean?
prompt = "Select File" : string?
filter = #f : (union false? regexp?)
filter-msg = "That filename does not have the right form." : string?
parent = (finder:dialog-parent-parameter) : (union (is-a?/c top-level-window<%>) false?)

This procedure queries the user for a single filename, using a platform-independent dialog box. Consider using `finder:put-file` instead of this function.

See section 10 for more information.

`finder:default-extension`

- `(finder:default-extension)` ⇒ string?
- `(finder:default-extension` *extension*`)` ⇒ void?
  extension : string?

  This parameter controls the default extension for the framework's `finder:put-file` dialog. Its value gets passed as the *default-extension* argument to `put-file`.

  Its default value is `""`.

`finder:default-filters`

- `(finder:default-filters` *filters*`)` ⇒ void?
  filters : (listof (list/p string? string?))
- `(finder:default-filters)` ⇒ (listof (list/p string? string?))

  This parameter controls the default extension for the framework's `finder:put-file` dialog. Its value gets passed as the *default-filters* argument to `put-file`.

  Its default value is `'(("Any" "*.*"))`.

`finder:dialog-parent-parameter`

- `(finder:dialog-parent-parameter` *parent*`)` ⇒ void?
  parent : (union false? (is-a?/c top-level-window<%>))
- `(finder:dialog-parent-parameter)` ⇒ (union false? (is-a?/c top-level-window<%>))

  This is a parameter (see parameters, §7.4 in *PLT MzScheme: Language Manual* for information about parameters) which determines the parent of the dialogs created by `finder:get-file`, `finder:put-file`, `finder:common-get-file`, `finder:common-put-file`, `finder:common-get-file-list`, `finder:std-get-file`, and `finder:std-put-file`.

`finder:get-file`

- `(finder:get-file` *directory prompt filter filter-msg parent*`)` ⇒ (union string? false?)
  directory = #f : (union string? false?)
  prompt = "Select File" : string?
  filter = #f : (union regexp? string? false?)
  filter-msg = "That filename does not have the right form." : string?
  parent = #f : (union false? (is-a?/c top-level-window<%>))

  Queries the user for a filename.

  If the result of `(preferences:get 'framework:file-dialogs)` is `'std` this calls `finder:std-get-file`, and if it is `'common`, `finder:common-get-file` is called.

`finder:put-file`

- (`finder:put-file` *name directory replace? prompt filter filter-msg parent*) ⇒ (union false? string?)
  name = "Untitled" : string?
  directory = #f : (union false? string?)
  replace? = #f : boolean?
  prompt = "Select File" : string?
  filter = #f : (union false? regexp?)
  filter-msg = "That filename does not have the right form." : string?
  parent = (finder:dialog-parent-parameter) : (union (is-a?/c `top-level-window<%>`) false?)

  Queries the user for a filename.

  If the result of (`preferences:get 'framework:file-dialogs`) is `'std` this calls `finder:std-put-file`, and if it is `'common`, `finder:common-put-file` is called.

`finder:std-get-file`

- (`finder:std-get-file` *directory prompt filter filter-msg parent*) ⇒ (union string? false?)
  directory = #f : (union string? false?)
  prompt = "Select File" : string?
  filter = #f : (union regexp? false?)
  filter-msg = "That filename does not have the right form." : string?
  parent = #f : (union false? (is-a?/c `top-level-window<%>`))

  This procedure queries the user for a single filename, using a platform-dependent dialog box. Consider using `finder:get-file` instead of this function.

  See section 10 for more information.

`finder:std-put-file`

- (`finder:std-put-file` *name directory replace? prompt filter filter-msg parent*) ⇒ (union false? string?)
  name = "Untitled" : string?
  directory = #f : (union false? string?)
  replace? = #f : boolean?
  prompt = "Select File" : string?
  filter = #f : (union false? regexp?)
  filter-msg = "That filename does not have the right form." : string?
  parent = (finder:dialog-parent-parameter) : (union (is-a?/c `top-level-window<%>`) false?)

  This procedure queries the user for a single filename, using a platform-dependent dialog box. Consider using `finder:put-file` instead of this function.

  See section 10 for more information.

`frame:add-snip-menu-items`

- (`frame:add-snip-menu-items` *menu menu-item%*) ⇒ void?
  menu : (is-a?/c `menu%`)
  menu-item% : (subclass?/c `menu-item%`)

  Inserts three menu items into *menu*, one that inserts a text box, one that inserts a pasteboard box, and one that inserts an image into the currently focused editor (if there is one). Uses *menu-item%* as the class for the menu items.

`frame:reorder-menus`

- (`frame:reorder-menus` *frame*) ⇒ void?
  frame : (is-a?/c `frame%`)

  Re-orders the menus in a frame. This is useful in conjunction with the `frame:standard-menus%` class. After instantiating that class and adding menus, the menus will be mis-ordered. This will put the File and Edit menus at the front of the menubar and the Help menu at the end.

`group:get-the-frame-group`

- (`group:get-the-frame-group`) ⇒ (is-a?/c `group:%`)

  This returns the frame group.

`gui-utils:cancel-on-right?`

- (`gui-utils:cancel-on-right?`) ⇒ boolean?

  Returns #t if cancel should be on the right-hand side (or below) in a dialog and #f otherwise.

  See also `gui-utils:ok/cancel-buttons`.

`gui-utils:cursor-delay`

- (`gui-utils:cursor-delay`) ⇒ real?

- (`gui-utils:cursor-delay` *new-delay*) ⇒ void?
  new-delay : real?

  This function is *not* a parameter. Instead, the state is just stored in the closure.

  The first case in the case lambda returns the current delay in seconds before a watch cursor is shown, when either `gui-utils:local-busy-cursor` or `gui-utils:show-busy-cursor` is called. The second case in the case lambda Sets the delay, in seconds, before a watch cursor is shown, when either `gui-utils:local-busy-cursor` or `gui-utils:show-busy-cursor` is called.

`gui-utils:delay-action`

- (`gui-utils:delay-action` *delay-time open close*) ⇒ void?
  delay-time : real?
  open : (-> void?)
  close : (-> void?)

  Use this function to delay an action for some period of time. It also supports cancelling the action before the time period elapses. For example, if you want to display a watch cursor, but you only want it to appear after 2 seconds and the action may or may not take more than two seconds, use this pattern:

```
(let ([close-down
       (gui-utils:delay-action
        2
        (lambda () .. init watch cursor ...)
        (lambda () .. close watch cursor ...))])
  ;; .. do action ...
  (close-down))
```

Creates a thread that waits *delay-time*. After *delay-time* has elapsed, if the result thunk has *not* been called, call *open*. Then, when the result thunk is called, call *close*. The function *close* will only be called if *open* has been called.

gui-utils:get-choice

- (gui-utils:get-choice *message* *true-choice* *false-choice* *title* *default-result* *paren*) ⇒ any?
  message : string?
  true-choice : string?
  false-choice : string?
  title = "Warning" : string?
  default-result = (quote disallow-close) : any?
  paren = #f : (union false? (is-a?/c frame%) (is-a?/c dialog%))

  Opens a dialog that presents a binary choice to the user. The user is forced to choose between these two options, ie cancelling or closing the dialog opens a message box asking the user to actually choose one of the two options.

  The dialog will contain the string *message* and two buttons, labeled with the *true-choice* and the *false-choice*. If the user clicks on *true-choice* `#t` is returned. If the user clicks on *false-choice*, `#f` is returned.

  The argument *default-result* determines how closing the window is treated. If the argument is `'disallow-close`, closing the window is not allowed. If it is anything else, that value is returned when the user closes the window.

  If gui-utils:cancel-on-right? returns #t, the false choice is on the right. Otherwise, the true choice is on the right.

gui-utils:get-clickback-delta

- (gui-utils:get-clickback-delta) ⇒ (is-a?/c style-delta%)

  This delta is designed for use with set-clickback. Use the result of this function as the style for the region text where the clickback is set.

  See also gui-utils:get-clicked-clickback-delta.

gui-utils:get-clicked-clickback-delta

- (gui-utils:get-clicked-clickback-delta) ⇒ (is-a?/c style-delta%)

  This delta is designed for use with set-clickback. Use it as one of the style-delta% argument to set-clickback.

  See also gui-utils:get-clickback-delta.

gui-utils:local-busy-cursor

- (gui-utils:local-busy-cursor *window* *thunk* *delay*) ⇒ any?
  window : (is-a?/c window<%>)
  thunk : (-> any?)
  delay = (gui-utils:cursor-delay) : integer?

  Evaluates (*thunk*) with a watch cursor in *window*. If *window* is `#f`, the watch cursor is turned on globally. The argument *delay* specifies the amount of time before the watch cursor is opened. Use gui-utils:cursor-delay to set this value for all uses of this function.

  The result of this function is the result of *thunk*.

gui-utils:next-untitled-name

- (gui-utils:next-untitled-name) ⇒ string?

  Returns a name for the next opened untitled frame. The first name is "Untitled", the second is "Untitled 2", the third is "Untitled 3", and so forth.

gui-utils:ok/cancel-buttons

- (gui-utils:ok/cancel-buttons *parent confirm-callback cancel-callback confirm-label cancel-label*) ⇒ (values (is-a?/c button%) (is-a?/c button%))
  parent : (is-a?/c area-container<%>)
  confirm-callback : ((is-a?/c button%) (is-a?/c event%) . -> . any)
  cancel-callback : ((is-a?/c button%) (is-a?/c event%) . -> . any)
  confirm-label = (string-constant ok) : string?
  cancel-label = (string-constant cancel) : string?

  Adds an Ok and a cancel button to a panel, changing the order to suit the platform. Under Mac OS and unix, the confirmation action is on the right (or bottom) and under Windows, the canceling action is on the right (or bottom). The confirmation action button has the '(border) style. The buttons are also sized to be the same width.

  The first result is be the OK button and the second is the cancel button.

  See also gui-utils:cancel-on-right?.

gui-utils:show-busy-cursor

- (gui-utils:show-busy-cursor *thunk delay*) ⇒ any?
  thunk : (-> any?)
  delay = (gui-utils:cursor-delay) : integer?

  Evaluates (*thunk*) with a watch cursor. The argument *delay* specifies the amount of time before the watch cursor is opened. Use gui-utils:cursor-delay to set this value to all calls.

  This function returns the result of *thunk*.

gui-utils:unsaved-warning

- (gui-utils:unsaved-warning *filename action can-save-now? parent*) ⇒ (symbols 'continue 'save 'cancel)
  filename : string?
  action : string?
  can-save-now? = #f : boolean?
  parent = #f : (union false? (is-a?/c frame%) (is-a?/c dialog%))

  This displays a dialog that warns the user of a unsaved file.

  The string, *action*, indicates what action is about to take place, without saving. For example, if the application is about to close a file, a good action is "Close Anyway". The result symbol indicates the user's choice. If *can-save-now?* is #f, this function does not give the user the "Save" option and thus will not return 'save.

handler:add-to-recent

- (handler:add-to-recent *filename*) ⇒ void?
  filename : string?

Adds a filename to the list of recently opened files.

`handler:current-create-new-window`

- (`handler:current-create-new-window` *new-window-handler*) ⇒ void
  new-window-handler : ((union false? string?) . -> . (is-a?/c `frame%`))

- (`handler:current-create-new-window`) ⇒ ((union false? string?) . -> . (is-a?/c `frame%`))

  This is a parameter that controls how the framework creates new application windows.

  The default setting is this:

```
(lambda (filename)
  (let ([frame (make-object frame:text-info-file% filename)])
    (send frame show #t)
    frame))
```

`handler:edit-file`

- (`handler:edit-file` *filename make-default*) ⇒ (union false? (is-a?/c `frame:editor<%>`))
  filename : (union string? false?)
  make-default = (lambda () ((handler:current-create-new-window) filename)) : (-> (is-a?/c `frame:editor<%>`))

  This function creates a frame or re-uses an existing frame to edit a file.

  If the preference 'framework:open-here is set to #t, and (send (`group:get-the-frame-group`) `get-open-here-frame`) returns a frame, the `open-here` method of that frame is used to load the file in the existing frame.

  Otherwise, it invokes the appropriate format handler to open the file (see `handler:insert-format-handler`).

  - If *filename* is a string, this function checks the result of `group:get-the-frame-group` to see if the *filename* is already open by a frame in the group.
    - If so, it returns the frame.
    - If not, this function calls `handler:find-format-handler` with *filename*.
      - If a handler is found, it is applied to *filename* and it's result is the final result.
      - If not, *make-default* is used.
  - If *filename* is `#f`, *make-default* is used.

`handler:find-format-handler`

- (`handler:find-format-handler` *filename*) ⇒ (string? . -> . (is-a?/c `frame:editor<%>`))
  filename : string?

  This function selects a format handler. See also `handler:insert-format-handler`.

  It finds a handler based on *filename*.

`handler:find-named-format-handler`

- (`handler:find-named-format-handler` *name*) ⇒ (string? . -> . (is-a?/c `frame:editor<%>`))
  name : string?

  This function selects a format handler. See also `handler:insert-format-handler`.

  It finds a handler based on *name*.

`handler:handler-extension`

- (`handler:handler-extension` *handler*) ⇒ string?
  handler : handler:handler?

  Extracts the extension from a handler.

`handler:handler-handler`

- (`handler:handler-handler` *handler*) ⇒ (string? . -> . (is-a?/c `frame:editor<%>`))
  handler : handler:handler?

  Extracs the handler's handling function

`handler:handler-name`

- (`handler:handler-name` *handler*) ⇒ string?
  handler : handler:handler?

  Extracts the name from a handler.

`handler:handler?`

- (`handler:handler?` *obj*) ⇒ boolean?
  obj : any?

  This predicate determines if its input is a handler

`handler:insert-format-handler`

- (`handler:insert-format-handler` *name* *pred* *handler*) ⇒ void?
  name : string?
  pred : (union string? (listof string?) (string? . -> . boolean?))
  handler : (string? . -> . (union false? (is-a?/c `frame:editor<%>`)))

  This function inserts a format handler.

  The string, *name* names the format handler for use with `handler:find-named-format-handler`. If *pred* is a string, it is matched with the extension of a filename by `handler:find-format-handler`. If *pred* is a list of strings, they are each matched with the extension of a filename by `handler:find-format-handler`. If it is a function, the filename is applied to the function and the functions result determines if this is the handler to use.

`handler:install-recent-items`

- (`handler:install-recent-items` *menu*) ⇒ void?
  menu : (is-a?/c `menu%`)

  This function deletes all of the items in the given menu and adds one menu item for each recently opened file. These menu items, when selected, call `handler:edit-file` with the filename of the recently opened file.

  The menu's size is limited to 10.

`handler:open-file`

- (`handler:open-file`) ⇒ (union false? (is-a?/c `frame:basic<%>`))

This function queries the user for a filename and opens the file for editing. It uses handler:edit-file to open the file, once the user has chosen it.

Calls finder:get-file and handler:edit-file.

handler:set-recent-items-frame-superclass

- (handler:set-recent-items-frame-superclass *frame*) ⇒ void?
  frame : (implementation?/c frame:standard-menus<%>)

Sets the superclass for the recently opened files frame. It must be derived from frame:standard-menus<%>.

handler:set-recent-position

- (handler:set-recent-position *filename start end*) ⇒ void?
  filename : string?
  start : number?
  end : number?

Sets the selection of the recently opened file to *start* and *end*.

handler:size-recently-opened-files

- (handler:size-recently-opened-files *num*) ⇒ void?
  num : number?

Sizes the 'framework:recently-opened-files/pos preference list length to *num*.

icon:get-anchor-bitmap

- (icon:get-anchor-bitmap) ⇒ (is-a?/c bitmap%)

This returns the anchor's bitmap%.

The bitmap may not respond \#t to the ok? method.

icon:get-autowrap-bitmap

- (icon:get-autowrap-bitmap) ⇒ (is-a?/c bitmap%)

This returns the autowrap's bitmap%.

The bitmap may not respond \#t to the ok? method.

icon:get-gc-off-bitmap

- (icon:get-gc-off-bitmap) ⇒ (is-a?/c bitmap%)

This returns a bitmap to be displayed in an frame:info<%> frame when garbage collection is *not* taking place.

The bitmap may not respond \#t to the ok? method.

icon:get-gc-on-bitmap

- (icon:get-gc-on-bitmap) ⇒ (is-a?/c bitmap%)

This returns a bitmap to be displayed in an frame:info<%> frame when garbage collection is taking place.

The bitmap may not respond \#t to the ok? method.

icon:get-left/right-cursor

- (icon:get-left/right-cursor) ⇒ (is-a?/c cursor%)

  This function returns a cursor% object that indicates left/right sizing is possible, for use with columns inside a window.

  The cursor may not respond \#t to the ok? method.

icon:get-lock-bitmap

- (icon:get-lock-bitmap) ⇒ (is-a?/c bitmap%)

  This returns the lock's bitmap%.

  The bitmap may not respond \#t to the ok? method.

icon:get-paren-highlight-bitmap

- (icon:get-paren-highlight-bitmap) ⇒ (is-a?/c bitmap%)

  This returns the parenthesis highlight bitmap%. It is only used on black and white screens.

icon:get-unlock-bitmap

- (icon:get-unlock-bitmap) ⇒ (is-a?/c bitmap%)

  This returns the reset unlocked bitmap%.

  The bitmap may not respond \#t to the ok? method.

icon:get-up/down-cursor

- (icon:get-up/down-cursor) ⇒ (is-a?/c cursor%)

  This function returns a cursor% object that indicates up/down sizing is possible, for use with columns inside a window.

  The cursor may not respond \#t to the ok? method.

keymap:add-to-right-button-menu

- (keymap:add-to-right-button-menu *func*) ⇒ void?
  func : ((is-a?/c popup-menu%) (is-a?/c editor<%>) (is-a?/c event%) . -> . void?)
- (keymap:add-to-right-button-menu) ⇒ ((is-a?/c popup-menu%) (is-a?/c editor<%>) (is-a?/c event%) . -> . void?)

  When the keymap that keymap:get-global returns is installed into an editor, this parameter's value is used for right button clicks.

  Before calling this procedure, the function append-editor-operation-menu-items is called.

  See also keymap:add-to-right-button-menu/before.

`keymap:add-to-right-button-menu/before`

- (`keymap:add-to-right-button-menu/before` *func*) ⇒ void?
  func : ((is-a?/c popup-menu%) (is-a?/c editor<%>) (is-a?/c event%) . -> . void?)
- (`keymap:add-to-right-button-menu/before`) ⇒ ((is-a?/c popup-menu%) (is-a?/c editor<%>) (is-a?/c event%) . -> . void?)

  When the keymap that `keymap:get-global` returns is installed into an editor, this function is called for right button clicks.

  After calling this procedure, the function `append-editor-operation-menu-items` is called.

  See also `keymap:add-to-right-button-menu`.

`keymap:call/text-keymap-initializer`

- (`keymap:call/text-keymap-initializer` *thunk-proc*) ⇒ any?
  thunk-proc : (-> any?)

  Thus function parameterizes the call to *thunk-proc* by setting the keymap-initialization procedure (see `current-text-keymap-initializer`) to install the framework's standard text bindings.

`keymap:canonicalize-keybinding-string`

- (`keymap:canonicalize-keybinding-string` *keybinding-string*) ⇒ string?
  keybinding-string : string?

  Returns a string that denotes the same keybindings as the input string, except that it is in canonical form; two canonical keybinding strings can be compared with `string=?`.

`keymap:get-editor`

- (`keymap:get-editor`) ⇒ (is-a?/c keymap%)

  This returns a keymap for handling standard editing operations. It binds these keys:

  - **z**: undo
  - **y**: redo
  - **x**: cut
  - **c**: copy
  - **v**: paste
  - **a**: select all

  where each key is prefixed with the menu-shortcut key, based on the platform. Under unix, the shortcut is scm"a:"; under windows the shortcut key is `"c:"` and under MacOS, the shortcut key is `"d:"`.

`keymap:get-file`

- (`keymap:get-file`) ⇒ (is-a?/c keymap%)

  This returns a keymap for handling file operations.

`keymap:get-global`

- (`keymap:get-global`) ⇒ (is-a?/c keymap%)

  This returns a keymap for general operations. See `keymap:setup-global` for a list of the bindings this keymap contains.

`keymap:get-search`

- `(keymap:get-search)` ⇒ (is-a?/c keymap%)

  This returns a keymap for searching operations

`keymap:make-meta-prefix-list`

- `(keymap:make-meta-prefix-list` *key*`)` ⇒ (listof string?)
  key : string?

  This prefixes a key with all of the different meta prefixes and returns a list of the prefixed strings.

  takes a keymap, a base key specification, and a function name; it prefixes the base key with all "meta" combination prefixes, and installs the new combinations into the keymap. For example, (`keymap:send-map-function-meta` *keymap* `"a"` *func*) maps all of "m:a" and "ESC;a" to *func*.

`keymap:send-map-function-meta`

- `(keymap:send-map-function-meta` *keymap key func*`)` ⇒ void?
  keymap : (is-a?/c keymap%)
  key : string?
  func : string?

  Most keyboard and mouse mappings are inserted into a keymap by calling the keymap's `map-function` method. However, "meta" combinations require special attention. The "m:" prefix recognized by `map-function` applies only to the Meta key that exists on some keyboards. By convention, however, "meta" combinations can also be accessed by using "ESC" as a prefix.

  This procedure binds all of the key-bindings obtained by prefixing *key* with a meta-prefix to *func* in *keymap*.

`keymap:setup-editor`

- `(keymap:setup-editor` *keymap*`)` ⇒ void?
  keymap : (is-a?/c keymap%)

  This sets up the input keymap with the bindings described in `keymap:get-editor`.

`keymap:setup-file`

- `(keymap:setup-file` *keymap*`)` ⇒ void?
  keymap : (is-a?/c keymap%)

  This extends a keymap% with the bindings for files.

`keymap:setup-global`

- `(keymap:setup-global` *keymap*`)` ⇒ void?
  keymap : (is-a?/c keymap%)

  This extends a keymap% with the general bindings.

  This function extends a keymap% with the following functions:

  – "ring-bell" (*any events*) — Rings the bell (using `bell`) and removes the search panel from the frame, if there.
  – "save-file" (*key events*) — Saves the buffer. If the buffer has no name, then `finder:put-file` is invoked.

- "save-file-as" (*key events*) — Calls `finder:put-file` to save the buffer.
- "load-file" (*key events*) — Invokes `finder:open-file`.
- "find-string" (*key events*) — Opens the search buffer at the bottom of the frame, unless it is already open, in which case it searches for the text in the search buffer.
- "find-string-reverse" (*key events*) — Same a "find-string", but in the reverse direction.
- "find-string-replace" (*key events*) — Opens a replace string dialog box.
- "toggle-anchor" (*key events*) — Turns selection-anchoring on or off.
- "center-view-on-line" (*key events*) — Centers the buffer in its display using the currently selected line.
- "collapse-space" (*key events*) — Collapses all non-return whitespace around the caret into a single space.
- "remove-space" (*key events*) — Removes all non-return whitespace around the caret.
- "collapse-newline" (*key events*) — Collapses all empty lines around the caret into a single empty line. If there is only one empty line, it is removed.
- "open-line" (*key events*) — Inserts a new line.
- "transpose-chars" (*key events*) — Transposes the characters before and after the caret and moves forward one position.
- "transpose-words" (*key events*) — Transposes words before and after the caret and moves forward one word.
- "capitalize-word" (*key events*) — Changes the first character of the next word to a capital letter and moves to the end of the word.
- "upcase-word" (*key events*) — Changes all characters of the next word to capital letters and moves to the end of the word.
- "downcase-word" (*key events*) — Changes all characters of the next word to lowercase letters and moves to the end of the word.
- "kill-word" (*key events*) — Kills the next word.
- "backward-kill-word" (*key events*) — Kills the previous word.
- "goto-line" (*any events*) — Queries the user for a line number and moves the caret there.
- "goto-position" (*any events*) — Queries the user for a position number and moves the caret there.
- "copy-clipboard" (*mouse events*) — Copies the current selection to the clipboard.
- "cut-clipboard" (*mouse events*) — Cuts the current selection to the clipboard.
- "paste-clipboard" (*mouse events*) — Patses the clipboard to the current selection.
- "copy-click-region" (*mouse events*) — Copies the region between the caret and the input mouse event.
- "cut-click-region" (*mouse events*) — Cuts the region between the caret and the input mouse event.
- "paste-click-region" (*mouse events*) — Pastes the clipboard into the position of the input mouse event.
- "select-click-word" (*mouse events*) — Selects the word under the input mouse event.
- "select-click-line" (*mouse events*) — Selects the line under the input mouse event.
- "start-macro" (*key events*) – Starts building a keyboard macro
- "end-macro" (*key events*) — Stops building a keyboard macro
- "do-macro" (*key events*) — Executes the last keyboard macro
- "toggle-overwrite" (*key events*) — Toggles overwriting mode

These functions are bound to the following keys (C = control, S = shift, A = alt, M = "meta", D = command):

- C-g : "ring-bell"
- M-C-g : "ring-bell"
- C-c C-g : "ring-bell"
- C-x C-g : "ring-bell"
- C-p : "previous-line"
- S-C-p : "select-previous-line"
- C-n : "next-line"
- S-C-n : "select-next-line"
- C-e : "end-of-line"
- S-C-e : "select-to-end-of-line"
- D-RIGHT : "end-of-line"
- S-D-RIGHT : "select-to-end-of-line"
- M-RIGHT : "end-of-line"
- S-M-RIGHT : "select-to-end-of-line"
- C-a : "beginning-of-line"
- S-C-a : "select-to-beginning-of-line"
- D-LEFT : "beginning-of-line"
- D-S-LEFT : "select-to-beginning-of-line"
- M-LEFT : "beginning-of-line"
- M-S-LEFT : "select-to-beginning-of-line"
- C-h : "delete-previous-character"
- C-d : "delete-next-character"
- C-f : "forward-character"
- S-C-f : "select-forward-character"
- C-b : "backward-character"
- S-C-b : "select-backward-character"
- M-f : "forward-word"

- S-M-f : “select-forward-word”
- A-RIGHT : “forward-word”
- A-S-RIGHT : “forward-select-word”
- M-b : “backward-word”
- S-M-b : “select-backward-word”
- A-LEFT : “backward-word”
- A-S-LEFT : “backward-select-word”
- M-d : “kill-word”
- M-DELETE : “backward-kill-word”
- M-c : “capitalize-word”
- M-u : “upcase-word”
- M-l : “downcase-word”
- M-< : “beginning-of-file”
- S-M-< : “select-to-beginning-of-file”
- M-> : “end-of-file”
- S-M-> : “select-to-end-of-file”
- C-v : “next-page”
- S-C-v : “select-next-page”
- M-v : “previous-page”
- S-M-v : “select-previous-page”
- C-l : “center-view-on-line”
- C-k : “delete-to-end-of-line”
- C-y : “paste-clipboard” (Except Windows)
- A-v : “paste-clipboard”
- D-v : “paste-clipboard”
- C-_ : “undo”
- C-x u : “undo”
- C-+ : “redo”
- C-w : “cut-clipboard”
- M-w : “copy-clipboard”
- C-x C-s : “save-file”
- C-x C-w : “save-file-as”
- C-x C-f : “load-file”
- C-s : “find-string”
- C-r : “find-string-reverse”
- M-% : “find-string-replace”
- SPACE : “collapse-space”
- M-\ : “remove-space”
- C-x C-o : “collapse-newline”
- C-o : “open-line”
- C-t : “transpose-chars”
- M-t : “transpose-words”
- C-SPACE : “toggle-anchor”
- M-g : “goto-line”
- M-p : “goto-position”
- LEFTBUTTONTRIPLE : “select-click-line”
- LEFTBUTTONDOUBLE : “select-click-word”
- RIGHTBUTTON : “copy-click-region”
- RIGHTBUTTONDOUBLE : “cut-click-region”
- MIDDLEBUTTON : “paste-click-region”
- C-RIGHTBUTTON : “copy-clipboard”
- INSERT : “toggle-overwrite”
- M-o : “toggle-overwrite”

`keymap:setup-search`

- (`keymap:setup-search` *keymap*) ⇒ void?
  keymap : (is-a?/c `keymap%`)

  This extends a `keymap%` with the bindings for searching.

`paren:backward-match`

- (`paren:backward-match` *text start end parens quotes comments containing? cache*) ⇒ (union false? (and/f integer? exact?))
  text : (is-a?/c `text%`)
  start : (and/f integer? exact?)
  end : (and/f integer? exact?)
  parens : (listof (cons/p string? string?))
  quotes : (listof (cons/p string? string?))
  comments : (listof string?)

containing? = #f : boolean?
cache = #f : (union false? (is-a?/c match-cache:%))

Returns the position in *text* that "opens" the text ending at *start*, or `#f` if no opening position is found (either because a parenthesis mis-match is discovered or the *end* boundary was reached). The match must occur before *end* (inclusive). Note that *start* > *end*, since *start* specifies the starting position of the search, not the earliest buffer position to be considered.

Spaces immediately preceding *start* are skipped. If the text at *start* is a close parenthesis or close quote, then the matching position is the opening parenthesis or quote. If a comment immediately precedes *start*, then the comment is skipped as whitespace. If an opening parenthesis immediately precedes *start*, then the matching position is *start* - 1. Otherwise, the matching position is the first whitespace or parenthesis character before *start*.

If *containing?* is not `#f`, then the matching procedure is modified as follows:

- Searching iterates backwards until some search fails. Then, the location of the last successful search is returned.
- If a mis-match is detected, then `#f` is returned.
- If there are no matches (and no mis-matches) before *start*, *start* itself is returned.

If *cache* is not `#f`, it must be an instance of `match-cache:%`. A cache object can be used to speed up successive calls to `paren:backward-match`. However, a buffer using a cache must call the cache's `invalidate` method when the buffer is modified. Different caches should be used for forward and backward matching. See section 17.1 for more information.

`paren:balanced?`

- (paren:balanced? *text start end parens quotes comments*) ⇒ boolean?
  text : (is-a?/c text%)
  start : (and/f integer? exact?)
  end : (and/f integer? exact?)
  parens : (listof (cons/p string? string?))
  quotes : (listof (cons/p string? string?))
  comments : (listof string?)

Returns `#t` if the text in *text* between positions *start* and *end* is *balanced*. The text is balanced if there are no unclosed parenthses or quotes, there are no closing parentheses that do not match an open parenthesis, and there are no mis-matched parentheses.

This uses `paren:forward-match`.

`paren:forward-match`

- (paren:forward-match *text start end parens quotes comments cache*) ⇒ (union false? (and/f integer? exact?))
  text : (is-a?/c text%)
  start : (and/f integer? exact?)
  end : (and/f integer? exact?)
  parens : (listof (cons/p string? string?))
  quotes : (listof (cons/p string? string?))
  comments : (listof string?)
  cache = #f : (union false? (is-a?/c match-cache:%))

This function returns the position in *text* that "closes" the text at *start*, or `#f` if no closing position is found (either because a parenthesis mis-match is discovered or the *end* boundary was reached). The match must occur before *end* (inclusive).

Spaces immediately following *start* are skipped. If the text at *start* is an open parenthesis or open quote, then the matching position is the closing parenthesis or quote. If a comment immediately follows *start*, it is skipped over as whitespace. If a closing parenthesis immediately follows *start* (after skipping whitespace), then `#f` is returned. Otherwise, the matching position is the position before the first whitespace, parenthesis, quote, or comment character after *start*.

If *cache* is not `#f`, it must be an instance of `match-cache:%`. A cache object can be used to speed up successive calls to `paren:forwardward-match`. However, a buffer using a cache must call the cache's `forward-invalidate` method when the buffer is modified. Different caches should be used for forward and backward matching. See section 17.1 for more information.

`paren:skip-whitespace`

- (`paren:skip-whitespace` *text pos dir*) ⇒ (and/f integer? exact?)
  text : (is-a?/c `text%`)
  pos : (and/f integer? exact?)
  dir : (symbols 'forward 'backward)

  If *dir* is `'forward`, this returns the position of the first non-whitespace character in *text* after *pos*. If *dir* is `'backward`, it returns the first non-whitespace character before *pos*.

`path-utils:generate-autosave-name`

- (`path-utils:generate-autosave-name` *filename*) ⇒ string?
  filename : string?

  Generates a name for an autosave file from *filename*.

`path-utils:generate-backup-name`

- (`path-utils:generate-backup-name` *filename*) ⇒ string?
  filename : string?

  Generates a name for an backup file from *filename*.

`preferences:add-callback`

- (`preferences:add-callback` *p f*) ⇒ (-> void?)
  p : symbol?
  f : (symbol? any? . -> . any?)

  This function adds a callback which is called with a symbol naming a preference and it's value, when the preference changes. `preferences:add-callback` returns a thunk, which when invoked, removes the callback from this preference.

  The callbacks will be called in the order in which they were added.

  If you are adding a callback for a preference that requires marshalling and unmarshalling, you must set the marshalling and unmarshalling functions by calling `preferences:set-un/marshall` before adding a callback.

  This function raises `exn:unknown-preference` if the preference has not been set.

`preferences:add-editor-checkbox-panel`

- (`preferences:add-editor-checkbox-panel`) ⇒ void?

  Adds a preferences panel for configuring options related to editing.

`preferences:add-font-panel`

- `(preferences:add-font-panel)` ⇒ void?

  Adds a font selection preferences panel to the preferences dialog.

`preferences:add-panel`

- `(preferences:add-panel` *name* *f*`)` ⇒ void?
  name : string?
  f : ((is-a?/c area-container-window<%>) . -> . (is-a?/c area-container-window<%>))

  `preferences:add-preference-panel` adds the result of *f* with name *name* to the preferences dialog box. When the preference dialog is opened for the first time, the function *f* is called with a panel, and *f* is expected to add a new child panel to it and add whatever preferences configuration controls it wants to that panel. Then, *f*'s should return the panel it added.

`preferences:add-scheme-checkbox-panel`

- `(preferences:add-scheme-checkbox-panel)` ⇒ void?

  Adds a preferences panel for configuring options related to Scheme.

`preferences:add-to-editor-checkbox-panel`

- `(preferences:add-to-editor-checkbox-panel` *proc*`)` ⇒ void?
  proc : ((is-a?/c vertical-panel%) . -> . void?)

  Saves *proc* until the preferences panel is created, when it is called with the Echeme preferences panel to add new children to the panel.

`preferences:add-to-scheme-checkbox-panel`

- `(preferences:add-to-scheme-checkbox-panel` *proc*`)` ⇒ void?
  proc : ((is-a?/c vertical-panel%) . -> . void?)

  Saves *proc* until the preferences panel is created, when it is called with the Scheme preferences panel to add new children to the panel.

`preferences:add-to-warnings-checkbox-panel`

- `(preferences:add-to-warnings-checkbox-panel` *proc*`)` ⇒ void?
  proc : ((is-a?/c vertical-panel%) . -> . void?)

  Saves *proc* until the preferences panel is created, when it is called with the Misc. panel to add new children to the panel.

`preferences:add-warnings-checkbox-panel`

- `(preferences:add-warnings-checkbox-panel)` ⇒ void?

  Adds a preferences panel for configuring options relating to warnings

`preferences:get`

- `(preferences:get` *symbol*`)` ⇒ any?
  symbol : symbol?

  See also `preferences:set-default`.

  `preferences:get` returns the value for the preference *symbol*. It raises `exn:unknown-preference` if the preference has not been set.

`preferences:hide-dialog`

- `(preferences:hide-dialog)` ⇒ void?

  Hides the preferences dialog.

`preferences:read`

- `(preferences:read)` ⇒ void?

  Reads the preferences from the preferences file and installs their values.

`preferences:restore-defaults`

- `(preferences:restore-defaults)` ⇒ void?

  `(preferences:restore-defaults)` restores the users's configuration to the default preferences.

`preferences:save`

- `(preferences:save)` ⇒ boolean?

  `(preferences:save-user-preferences)` saves the user's preferences to disk, potentially marshalling some of the preferences.

  Returns `\#f` if saving the preferences fails and `\#t` otherwise.

`preferences:set`

- `(preferences:set` *symbol value*`)` ⇒ void?
  symbol : symbol?
  value : any?

  `preferences:set-preference` sets the preference *symbol* to *value*. This should be called when the users requests a change to a preference.

`preferences:set-default`

- `(preferences:set-default` *symbol value test*`)` ⇒ void?
  symbol : symbol?
  value : any?
  test : (any? . -> . any?)

  This function must be called every time your application starts up, before any call to `preferences:get`.

  If you use `preferences:set-un/marshall`, you must also call it before calling this function.

  This sets the default value of the preference *symbol* to *value*. If the user has chosen a different setting, the user's setting will take precedence over the default value.

The last argument, *test* is used as a safeguard. That function is called to determine if a preference read in from a file is a valid preference. If *test* returns `#t`, then the preference is treated as valid. If *test* returns `#f` then the default is used. If there is a site-wide default preferences file, the default preference in that file is used instead of *value*.

`preferences:set-un/marshall`

- (`preferences:set-un/marshall` *symbol marshall unmarshall*) ⇒ void?
  symbol : symbol?
  marshall : (any? . -> . printable?)
  unmarshall : (printable? . -> . any?)

  `preference:set-un/marshall` is used to specify marshalling and unmarshalling functions for the preference *symbol*. *marshall* will be called when the users saves their preferences to turn the preference value for *symbol* into a printable value. *unmarshall* will be called when the user's preferences are read from the file to transform the printable value into it's internal representation. If `preference:set-un/marshall` is never called for a particular preference, the values of that preference are assumed to be printable.

  If the unmarshalling function returns a value that does not meet the guard passed to `preferences:set-default` for this preference, the default value is used.

  `preference:set-un/marshall` must be called before calling `preferences:get` or `preferences:set-default`.

`preferences:show-dialog`

- (`preferences:show-dialog`) ⇒ void?

  Shows the preferences dialog.

`scheme-paren:backward-containing-sexp`

- (`scheme-paren:backward-containing-sexp` *text start end cache*) ⇒ (union false? (and/f integer? exact?))
  text : (is-a?/c `text%`)
  start : (and/f integer? exact?)
  end : (and/f integer? exact?)
  cache = #f : (union false? (is-a?/c `match-cache:%`))

  Returns the beginning of the interior of the (non-atomic) S-expression containing *start*.

`scheme-paren:backward-match`

- (`scheme-paren:backward-match` *text start end cache*) ⇒ (union false? (and/f integer? exact?))
  text : (is-a?/c `text%`)
  start : (and/f integer? exact?)
  end : (and/f integer? exact?)
  cache = #f : (union false? (is-a?/c `match-cache:%`))

  Specializes `paren:backward-match` to Scheme.

`scheme-paren:balanced?`

- (`scheme-paren:balanced?` *text start end*) ⇒ boolean?
  text : (is-a?/c `text%`)

start : (and/f integer? exact?)
end : (and/f integer? exact?)

Specializes paren:balanced? to Scheme.

scheme-paren:forward-match

- (scheme-paren:forward-match *text start end cache*) ⇒ (union false? (and/f integer? exact?))
  text : (is-a?/c text%)
  start : (and/f integer? exact?)
  end : (and/f integer? exact?)
  cache = #f : (union false? (is-a?/c match-cache:%))

  Specializes paren:forward-match to Scheme.

scheme-paren:get-comments

- (scheme-paren:get-comments) ⇒ (listof string?)

  Returns the comment characters for Scheme.

scheme-paren:get-paren-pairs

- (scheme-paren:get-paren-pairs) ⇒ (listof (cons/p string? string?))

  Returns the paren pairs for Scheme.

scheme-paren:get-quote-pairs

- (scheme-paren:get-quote-pairs) ⇒ (listof (cons/p string? string?))

  Returns the quote pairs for Scheme.

scheme:add-preferences-panel

- (scheme:add-preferences-panel) ⇒ void?

  Adds a tabbing preferences panel to the preferences dialog.

scheme:get-keymap

- (scheme:get-keymap) ⇒ (is-a?/c keymap%)

  Returns a keymap with binding suitable for Scheme.

scheme:get-style-list

- (scheme:get-style-list) ⇒ (is-a?/c style-list%)

  Returns a style list that is used for all Scheme buffers.

scheme:get-wordbreak-map

- (scheme:get-wordbreak-map) ⇒ (is-a?/c editor-wordbreak-map%)

  This method returns a editor-wordbreak-map% that is suitable for Scheme.

`scheme:init-wordbreak-map`

- (`scheme:init-wordbreak-map` *key*) ⇒ void?
  key : (is-a?/c keymap%)

  Initializes the workdbreak map for *keymap*.

`scheme:setup-keymap`

- (`scheme:setup-keymap` *keymap*) ⇒ void?
  keymap : (is-a?/c keymap%)

  Initializes *keymap* with Scheme-mode keybindings.

`test:button-push`

- (`test:button-push` *button*) ⇒ void?
  button : (union (and/f string? (lambda (str) (test:top-level-focus-window-has? (lambda (c) (and (is-a? c button%) (string=? (send c get-label) str) (send c is-enabled?) (send c is-shown?)))))) (and/f (is-a?/c button%) (lambda (btn) (and (send btn is-enabled?) (send btn is-shown?))) (lambda (btn) (test:top-level-focus-window-has? (lambda (c) (eq? c btn))))))

  Simulates pushing *button*. If a string is supplied, the primitive searches for a button labelled with that string in the active frame. Otherwise, it pushes the button argument.

`test:close-top-level-window`

- (`test:close-top-level-window` *tlw*) ⇒ void?
  tlw : (is-a?/c top-level-window<%>)

  Use this function to simulate clicking on the close box of a frame. Closes *tlw* with this expression:

```
(when (send tlw can-close?)
  (send tlw on-close)
  (send tlw show #f))
```

`test:current-get-eventspaces`

- (`test:current-get-eventspaces` *func*) ⇒ void?
  func : (-> (listof eventspace?))
- (`test:current-get-eventspaces`) ⇒ (-> (listof eventspace?))

  This parameter that specifies which eventspace (see section 2.4) are considered when finding the frontmost frame. The first case sets the parameter to *func*. The procedure *func* will be invoked with no arguments to determine the eventspaces to consider when finding the frontmost frame for simulated user events. The second case returns the current value of the parameter. This will be a procedure which, when invoked, returns a list of eventspaces.

`test:keystroke`

- (`test:keystroke` *key* *modifier-list*) ⇒ void?
  key : (union char? symbol?)
  modifier-list = null : (listof (symbols (quote alt) (quote control) (quote meta) (quote shift) (quote noalt) (quote nocontrol) (quote nometea) (quote noshift)))

This function simulates a user pressing a key. The argument, *key*, is just like the argument to the `get-key-code` method of the `key-event%` class.

Note: To send the "Enter" key, use `# eturn`, not `# ewline`.

The `'shift` or `'noshift` modifier is implicitly set from *key*, but is overridden by the argument list. The `'shift` modifier is set for any capitol alpha-numeric letters and any of the following characters:

```
#\? #\: #\~ #\\ #\|
#\< #\> #\{ #\} #\[ #\] #\( #\)
#\! #\@ #\# #\$ #\% #\^ #\& #\* #\_ #\+
```

If conflicting modifiers are provided, the ones later in the list are used.

test:menu-select

- (test:menu-select *menu* *item*) ⇒ void?
  menu : string?
  item : string?

Selects the menu-item named *item* in the menu named *menu*.

Note: The string for the menu item does not include its keyboard equivalent. For example, to select "New" from the "File" menu, use "New", not "New Ctrl+m n".

test:mouse-click

- (test:mouse-click *button* *x* *y* *modifiers*) ⇒ void?
  button : (symbols 'left 'middle 'right)
  x : (and/f exact? integer?)
  y : (and/f exact? integer?)
  modifiers = null : (listof (symbols (quote alt) (quote control) (quote meta) (quote shift) (quote noalt) (quote nocontrol) (quote nometa) (quote noshift)))

Simulates a mouse click at the coordinate: (x, y) in the currently focused `window<%>`, assuming that it supports the `on-event` method. Use `test:button-push` to click on a button.

On the Macintosh, `'right` corresponds to holding down the command modifier key while clicking and `'middle` cannot be generated.

Under Windows, `'middle` can only be generated if the user has a three button mouse.

The modifiers later in the list *modifiers* take precedence over ones that appear earlier.

test:new-window

- (test:new-window *window*) ⇒ void?
  window : (is-a?/c `window<%>`)

Moves the keyboard focus to a new window within the currently active frame. Unfortunately, neither this function nor any other function in the test engine can cause the focus to move from the top-most (active) frame.

test:number-pending-actions

- (test:number-pending-actions) ⇒ number?

Returns the number of pending events (those that haven't completed yet)

`test:reraise-error`

- `(test:reraise-error)` ⇒ void?

  See also Errors, section 3.3.3.

`test:run-interval`

- `(test:run-interval` *msec*`)` ⇒ void?
  msec : number?

- `(test:run-interval)` ⇒ number?

  See also Actions and completeness, section 3.3.2. The first case in the case-lambda sets the run interval to *msec* milliseconds and the second returns the current setting.

`test:run-one`

- `(test:run-one` *f*`)` ⇒ void?
  f : (-> void?)

  Runs the function *f* as if it was a simulated event. See also section 3.3.

`test:set-check-box!`

- `(test:set-check-box!` *check-box state*`)` ⇒ void?
  check-box : (union string? (is-a?/c `check-box%`))
  state : boolean?

  Clears the `check-box%` item if *state* is `#f`, and sets it otherwise.

  If *check-box* is a string, this function searches for a `check-box%` with a label matching that string, otherwise it uses *check-box* itself.

`test:set-choice!`

- `(test:set-choice!` *choice str*`)` ⇒ void?
  choice : (union string? (is-a?/c `choice%`))
  str : string?

  Selects *choice*'s item *str*. If *choice* is a string, this function searches for a `choice%` with a label matching that string, otherwise it uses *choice* itself.

`test:set-radio-box!`

- `(test:set-radio-box!` *radio-box state*`)` ⇒ void?
  radio-box : (union string? (is-a?/c `radio-box%`))
  state : (union string? number?)

  Sets the radio-box to *state*. If *state* is a string, this function finds the choice with that label and if it is a number, it uses the number as an index into the state. If the number is out of range or if the label isn't in the radio box, an exception is raised.

  If *radio-box* is a string, this function searches for a `radio-box%` with a label matching that string, otherwise it uses *radio-box* itself.

`test:set-radio-box-item!`

- `(test:set-radio-box-item!` *entry*`)` ⇒ void?
  entry : string?

  Finds a `radio-box%` that has a label *entry* and sets the radio-box to *entry*.

`test:top-level-focus-window-has?`

- `(test:top-level-focus-window-has?` *test*`)` ⇒ boolean?
  test : ((is-a?/c `area<%>`) . -> . boolean?)

  Calls *test* for each child of the top-level-focus-frame and returns #t if *test* ever does, otherwise returns #f. If there is no top-level-focus-window, returns #f.

`version:add-spec`

- `(version:add-spec` *spec* *revision*`)` ⇒ void?
  spec : any?
  revision : any?

  These two values are appended to the version string. `write` is used to transform them to strings. For example:

  ```
  (version:add-version-spec 's 1)
  ```

  in version 201 will make the version string be `"201s1"`. The symbols `'f` and `'d` are used internally for framework and drscheme revisions.

`version:version`

- `(version:version)` ⇒ string?

  This function returns a string describing the version of this application. See also `version:add-spec`.

Index