Framework: Racket GUI Application Framework

Version 5.0

Robert Bruce Findler
and Matthew Flatt

June 6, 2010

```
(require framework)
```

The framework provides a number of mixins, classes and functions designed to help you build a complete application program on top of the `racket/gui` library.

- **Entire Framework**

  - `(require framework)`

    This library provides all of the definitions and syntax described in this manual.

  - `(require framework/framework-sig)`

    This library provides the signature definitions: `framework^`, and `framework-class^`. The `framework^` signature contains 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 framework/framework-unit)`

    This library provides one `unit/sig`: `framework@`. It exports the signature `framework^`. It imports the `mred^` signature.

- **Test Suite Engine**

  `(require framework/test)`

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

- **GUI Utilities** `(require framework/gui-utils)`

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

- **Preferences** (`require` `framework/preferences`)

  This library provides a subset of the names of the `framework.ss` library, namely those for manipulating preference settings and is designed to be used from `racket`.

  The precise set of exported names is: `preferences:snapshot?`, `preferences:restore-prefs-snapshot`, `preferences:get-prefs-snapshot`, `exn:make-unknown-preference`, `exn:unknown-preference?`, `preferences:low-level-put-preferences`, `preferences:get`, `preferences:set`, `preferences:add-callback`, `preferences:set-default`, `preferences:set-un/marshall`, and `preferences:restore-defaults`.

- **Decorated Editor Snip** (`require` `framework/decorated-editor-snip`)

  This library is here for backwards compatibility. The functionality in it has moved into the framework proper, in the §8 "Editor Snip" section.

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.

1 Application

```
(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*.

2   Autosave

`autosave:autosavable<%>` : `interface?`

Classes that implement this interface can be autosaved.

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

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

(`autosave:register` *obj*) → `void?`
  obj : (`and/c` (`is-a?/c` `autosave:autosavable<%>`)
               (`is-a?/c` `editor<%>`)))

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.

(`autosave:restore-autosave-files/gui`) → `void?`

Opens a GUI to ask the user about recovering any autosave files left around from crashes and things.

This function doesn't return until the user has finished restoring the autosave files. (It uses yield to handle events however.)

3 Canvas

`canvas:basic<%>` : `interface?`
  implements: `editor-canvas%`

`canvas:basic-mixin` : `(class? . -> . class?)`
  argument extends/implements: `editor-canvas%`
  result implements: `canvas:basic<%>`

`canvas:color<%>` : `interface?`
  implements: `canvas:basic<%>`

Mixins that implement this interface initialize the background color of the canvas to the value of the `'framework:basic-canvas-background` preference. Adds a callback so that when that preference is modified, the background color changes.

`canvas:color-mixin` : `(class? . -> . class?)`
  argument extends/implements: `canvas:basic<%>`
  result implements: `canvas:color<%>`

`canvas:delegate<%>` : `interface?`
  implements: `canvas:basic<%>`

This class is part of the delegate window implementation.

`canvas:delegate-mixin` : `(class? . -> . class?)`
  argument extends/implements: `canvas:basic<%>`

result implements: `canvas:delegate<%>`

Provides an implementation of `canvas:delegate<%>`.

`(send a-canvas:delegate on-superwindow-show shown?)` → `void`
  `shown?` : `boolean?`

>   Overrides `on-superwindow-show` in `window<%>`.

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

`canvas:info<%>` : `interface?`
  implements: `canvas:basic<%>`

`canvas:info-mixin` : `(class? . -> . class?)`
  argument extends/implements: `canvas:basic<%>`
  result implements: `canvas:info<%>`

`(send a-canvas:info on-focus)` → `void`

>   Overrides `on-focus` in `editor-canvas%`.

>   sets the canvas that the frame displays info about.

`(send a-canvas:info set-editor)` → `void`

>   Overrides `set-editor` in `editor-canvas%`.

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

`canvas:wide-snip<%>` : `interface?`
  implements: `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.

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

Recalculates the sizes of the wide snips.

`(send a-canvas:wide-snip add-wide-snip snip) → void`
  `snip : (instance 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-tall-snip snip) → void`
  `snip : (instance 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.

`canvas:wide-snip-mixin : (class? . -> . class?)`
  argument extends/implements: `canvas:basic<%>`
  result implements: `canvas:wide-snip<%>`

This canvas maintains 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.

`(send a-canvas:wide-snip on-size width`
                                     `height) → void`
  `width : (integer-in 0 10000)`
  `height : (integer-in 0 10000)`

Overrides `on-size` in `editor-canvas%`.

Adjusts the sizes of the marked snips.

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

`canvas:basic% : class?`
  superclass: `(canvas:basic-mixin editor-canvas%)`

`canvas:color% : class?`
  superclass: `(canvas:color-mixin canvas:basic%)`

```
canvas:info% : class?
  superclass: (canvas:info-mixin canvas:basic%)
```

```
canvas:delegate% : class?
  superclass: (canvas:delegate-mixin canvas:basic%)
```

```
canvas:wide-snip% : class?
  superclass: (canvas:wide-snip-mixin canvas:basic%)
```

4   Color Model

```
(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 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 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 x y z) → (list/c 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? val) → boolean?
  val : any/c
```

Determines if `val` an xyz color record.

`(color-model:xyz-x xyz)` → `number?`
  `xyz` : `color-model:xyz?`

Extracts the x component of `xyz`.

`(color-model:xyz-y xyz)` → `number?`
  `xyz` : `color-model:xyz?`

Extracts the y component of `xyz`.

`(color-model:xyz-z xyz)` → `number?`
  `xyz` : `color-model:xyz?`

Extracts the z component of `xyz`.

5   Color Prefs

```
(color-prefs:set-default/color-scheme pref-sym
                                      black-on-white-color
                                      white-on-black-color)
 → void?
  pref-sym : symbol?
  black-on-white-color : (or/c (is-a?/c color%) string?)
  white-on-black-color : (or/c (is-a?/c color%) string?)
```

Registers a preference whose value will be updated when the user clicks on one of the color scheme default settings in the preferences dialog.

Also calls `preferences:set-default` and `preferences:set-un/marshall` with appropriate arguments to register the preference.

```
(color-prefs:register-color-preference  pref-name
                                        style-name
                                        color/sd
                                       [white-on-black-color])
 → void?
  pref-name : symbol?
  style-name : string?
  color/sd : (or/c (is-a?/c color%) (is-a?/c style-delta%))
  white-on-black-color : (or/c string? (is-a?/c color%) false/c)
                       = #f
```

This function registers a color preference and initializes the style list returned from `editor:get-standard-style-list`. In particular, it calls `preferences:set-default` and `preferences:set-un/marshall` to install the pref for *pref-name*, using *color/sd* as the default color. The preference is bound to a `style-delta%`, and initially the `style-delta%` changes the foreground color to *color/sd*, unless *color/sd* is a style delta already, in which case it is just used directly. Then, it calls `editor:set-standard-style-list-delta` passing the *style-name* and the current value of the preference *pref-name*.

Finally, it adds calls `preferences:add-callback` to set a callback for *pref-name* that updates the style list when the preference changes.

If *white-on-black-color* is not `#f`, then the color of the *color/sd* argument is used in combination with *white-on-black-color* to register this preference with `color-prefs:set-default/color-scheme`.

```
(color-prefs:add-background-preferences-panel) → void?
```

11

Adds a preferences panel that configures the background color for `editor:basic-mixin`.

```
(color-prefs:add-to-preferences-panel name
                                       func) → void?
  name : string?
  func : ((is-a?/c vertical-panel%) . -> . void?)
```

Calls *func* with the subpanel of the preferences coloring panel that corresponds to *name*.

```
(color-prefs:build-color-selection-panel parent
                                          pref-sym
                                          style-name
                                          example-text) → void?
  parent : (is-a?/c area-container<%>)
  pref-sym : symbol?
  style-name : string?
  example-text : string?
```

Builds a panel with a number of controls for configuring a font: the color and check boxes for bold, italic, and underline. The *parent* argument specifies where the panel will be placed. The *pref-sym* should be a preference (suitable for use with `preferences:get` and `preferences:set`). The *style-name* specifies the name of a style in the style list returned from `editor:get-standard-style-list` and *example-text* is shown in the panel so users can see the results of their configuration.

```
(color-prefs:marshall-style-delta style-delta) → printable/c
  style-delta : (is-a?/c style-delta%)
```

Builds a printed representation for a style-delta.

```
(color-prefs:unmarshall-style-delta marshalled-style-delta)
 → (or/c false/c (is-a?/c style-delta%))
  marshalled-style-delta : printable/c
```

Builds a style delta from its printed representation. Returns `#f` if the printed form cannot be parsed.

```
(color-prefs:white-on-black) → any
```

Sets the colors registered by `color-prefs:register-color-preference` to their white-on-black variety.

`(color-prefs:black-on-white)` → `any`

Sets the colors registered by `color-prefs:register-color-preference` to their black-on-white variety.

6   Color

---

color:text<%> : interface?
  implements: text:basic<%>

This interface describes how coloring is stopped and started for text that knows how to color itself. It also describes how to query the lexical and s-expression structure of the text.

```
(send a-color:text start-colorer token-sym->style
                                 get-token
                                 pairs)              → void
  token-sym->style : (-> symbol? string?)
  get-token : (or/c (-> input-port?
                        (values any/c
                                symbol?
                                (or/c false? symbol?)
                                exact-nonnegative-integer?
                                exact-nonnegative-integer?))
                    (-> input-port?
                        exact-nonnegative-integer?
                        any/c
                        (values any/c
                                symbol?
                                (or/c false? symbol?)
                                exact-nonnegative-integer?
                                exact-nonnegative-integer?
                                exact-nonnegative-integer?
                                any/c)))
  pairs : (listof (list/p symbol? symbol?))
```

    Starts tokenizing the buffer for coloring and parenthesis matching.

    The `token-sym->style` argument will be passed the first return symbol from `get-token`, and it should return the style-name that the token should be colored.

    The `get-token` argument takes an input port and optionally an offset and mode value. When it accepts just an input port, `get-token` returns the next token as 5 values:

- An unused value. This value is intended to represent the textual component of the token and may be used as such in the future.

- A symbol describing the type of the token. This symbol is transformed into a style-name via the `token-sym->style` argument. The symbols `'white-space` and `'comment` have special meaning and should always

be returned for white space and comment tokens respectively. The symbol `'no-color` can be used to indicate that although the token is not white space, it should not be colored. The symbol `'eof` must be used to indicate when all the tokens have been consumed.

- A symbol indicating how the token should be treated by the paren matcher or `#f`. This symbol should be in the pairs argument.
- The starting position of the token.
- The ending position of the token.

When `get-token` accepts an offset and mode value in addition to an input port, it must also return two extra results, which are a backup distance and new mode. The offset given to `get-token` can be added to the position of the input port to obtain absolute coordinates within a text stream. The mode argument allows `get-token` to communicate information from earlier parsing to later. When `get-token` is called for the beginning on a stream, the mode argument is `#f`; thereafter, the mode returned for the previous token is provided to `get-token` for the next token. The mode should not be a mutable value; if part of the stream is re-tokenized, the mode saved from the immediately preceding token is given again to the `get-token` function. The backup distance returned by `get-token` indicates the maximum number of characters to back up (counting from the start of the token) and for re-parsing after a change to the editor within the token's region.

The `get-token` function is usually be implemented with a lexer using the `parser-tools/lex` library. The `get-token` function must obey the following invariants:

- Every position in the buffer must be accounted for in exactly one token, and every token must have a non-zero width.
- The token returned by `get-token` must rely only on the contents of the input port argument plus the mode argument. This constraint means that the tokenization of some part of the input cannot depend on earlier parts of the input except through the mode (and implicitly through the starting positions for tokens).
- A change to the stream must not change the tokenization of the stream prior to the token immediately preceding the change plus the backup distance. In the following example, this invariant does not hold for a zero backup distance: If the buffer contains

  ```
  " 1 2 3
  ```

  and the tokenizer treats the unmatched " as its own token (a string error token), and separately tokenizes the 1 2 and 3, an edit to make the buffer look like

  ```
  " 1 2 3"
  ```

  would result in a single string token modifying previous tokens. To handle these situations, `get-token` can treat the first line as a single token, or it can precisely track backup distances.

The *pairs* argument is a list of different kinds of matching parens. The second value returned by *get-token* is compared to this list to see how the paren matcher should treat the token. An example: Suppose pairs is '((|(| |)|) (|[| |]|) (begin end)). This means that there are three kinds of parens. Any token which has 'begin as its second return value will act as an open for matching tokens with 'end. Similarly any token with '|]| will act as a closing match for tokens with '|[|. When trying to correct a mismatched closing parenthesis, each closing symbol in pairs will be converted to a string and tried as a closing parenthesis.

(send *a-color:text* stop-colorer [*clear-colors*]) → void
  clear-colors : boolean = #t

> Stops coloring and paren matching the buffer.

> If *clear-colors* is true all the text in the buffer will have its style set to Standard.

(send *a-color:text* force-stop-colorer *stop?*) → void
  stop? : boolean?

> Causes the entire tokenizing/coloring system to become inactive. Intended for debugging purposes only.

> *stop?* determines whether the system is being forced to stop or allowed to wake back up.

(send *a-color:text* is-stopped?) → boolean?

> Indicates if the colorer for this editor has been stopped, or not.

(send *a-color:text* is-frozen?) → boolean?

> Indicates if this editor's colorer is frozen. See also freeze-colorer and thaw-colorer.

(send *a-color:text* freeze-colorer) → void

> Keep the text tokenized and paren matched, but stop altering the colors.

> freeze-colorer will not return until the coloring/tokenization of the entire text is brought up-to-date. It must not be called on a locked text.

(send *a-color:text* thaw-colorer [*recolor*
                                    retokenize]) → void
  recolor : boolean = #t
  retokenize : boolean = #f

> Start coloring a frozen buffer again.

> If recolor? is #t, the text is re-colored. If it is #f the text is not recolored. When recolor? is #t, retokenize? controls how the text is recolored. #f

16

causes the text to be entirely re-colored before thaw-colorer returns using the existing tokenization. `#t` causes the entire text to be retokenized and recolored from scratch. This will happen in the background after the call to `thaw-colorer` returns.

```
(send a-color:text reset-region start end) → void
  start : natural-number?
  end : (union 'end  natural-number?)
```

Set the region of the text that is tokenized.

```
(send a-color:text reset-regions regions) → void
  regions : (listof (list number (union 'end  number)))
```

Sets the currently active regions to be *regions*.

```
(send a-color:text get-regions)
 → (listof (list number (union 'end  number)))
```

This returns the list of regions that are currently being colored in the editor.

```
(send a-color:text skip-whitespace position
                                   direction
                                   comments?) → natural-number?
  position : natural-number?
  direction : (symbols 'forward  'backward)
  comments? : boolean?
```

Returns the next non-whitespace character.

Starts from position and skips whitespace in the direction indicated by direction. If *comments?* is true, comments are skipped as well as whitespace. skip-whitespace determines whitespaces and comments by comparing the token type to `'white-space` and `'comment`.

Must only be called while the tokenizer is started.

```
(send a-color:text backward-match position
                                   cutoff)
 → (union natural-number? false?)
  position : natural-number?
  cutoff : natural-number?
```

Skip all consecutive whitespaces and comments (using `skip-whitespace`) immediately preceding the position. If the token at this position is a close, return the position of the matching open, or `#f` if there is none. If the token was an open, return `#f`. For any other token, return the start of that token.

Must only be called while the tokenizer is started.

```
(send a-color:text backward-containing-sexp position
                                              cutoff)
 → (union natural-number? false?)
  position : natural-number?
  cutoff : natural-number?
```

Return the starting position of the interior of the (non-atomic) s-expression containing position, or #f is there is none.

Must only be called while the tokenizer is started.

```
(send a-color:text forward-match position
                                  cutoff)
 → (union natural-number? false?)
  position : natural-number?
  cutoff : natural-number?
```

Skip all consecutive whitespaces and comments (using skip-whitespace) immediately following position. If the token at this position is an open, return the position of the matching close, or #f if there is none. For any other token, return the end of that token.

Must only be called while the tokenizer is started.

```
(send a-color:text insert-close-paren position
                                       char
                                       flash?
                                       fixup?)   → void
  position : natural-number?
  char : char?
  flash? : boolean?
  fixup? : boolean?
```

The position is the place to put the parenthesis, and char is the parenthesis to be added (e.g., that the user typed). If fixup? is true, the right kind of closing parenthesis will be chosen from the set previously passed to start-colorer—but only if an inserted char would be colored as a parenthesis (i.e., with the 'parenthesis classification). Otherwise, char will be inserted, even if it is not the right kind. If flash? is true, the matching open parenthesis will be flashed.

```
(send a-color:text classify-position position) → symbol?
  position : natural-number?
```

Return a symbol for the lexer-determined token type for the token that contains the item after position.

Must only be called while the tokenizer is started.

`color:text-mixin` : `(class? . -> . class?)`
   argument extends/implements: `text:basic<%>`
   result implements: `color:text<%>`

Adds the functionality needed for on-the-fly coloring and parenthesis matching based on incremental tokenization of the text.

`(send` *a-color:text* `lock)` → `void`
      Overrides `lock` in `editor<%>`.

`(send` *a-color:text* `on-focus)` → `void`
      Overrides `on-focus` in `editor<%>`.

`(send` *a-color:text* `after-edit-sequence)` → `void`
      Augments `after-edit-sequence` in `editor<%>`.

`(send` *a-color:text* `after-set-position)` → `void`
      Augments `after-set-position` in `text%`.

`(send` *a-color:text* `after-change-style)` → `void`
      Augments `after-change-style` in `text%`.

`(send` *a-color:text* `on-set-size-constraint)` → `void`
      Augments `on-set-size-constraint` in `text%`.

`(send` *a-color:text* `after-insert)` → `void`
      Augments `after-insert` in `text%`.

`(send` *a-color:text* `after-delete)` → `void`
      Augments `after-delete` in `text%`.

`color:text%` : `class?`
   superclass: `(color:text-mixin text:keymap%)`

`color:text-mode<%>` : `interface?`

`color:text-mode-mixin` : `(class? . -> . class?)`
  argument extends/implements: `mode:surrogate-text<%>`
  result implements: `color:text-mode<%>`

This mixin adds coloring functionality to the mode.

```
(new color:text-mode-mixin
 [[get-token get-token]
  [token-sym->style token-sym->style]
  [matches matches]])
→ (is-a?/c color:text-mode-mixin)
 get-token : lexer = default-lexer
 token-sym->style :  (symbol? . -> . string?)
                  = (λ (x) "Standard")
 matches : (listof (list/c symbol? symbol?)) = null
```

The arguments are passed to `start-colorer`.

`(send a-color:text-mode on-disable-surrogate)` → `void`

Overrides `on-disable-surrogate` in `mode:surrogate-text<%>`.

`(send a-color:text-mode on-enable-surrogate)` → `void`

Overrides `on-enable-surrogate` in `mode:surrogate-text<%>`.

`color:text-mode%` : `class?`
  superclass: `(color:text-mode-mixin mode:surrogate-text%)`

7 Comment Box

`comment-box:snip%` : `class?`
  superclass: `editor-snip:decorated%`
  extends: `readable-snip<%>`

This snip implements the comment boxes that you see in DrRacket.

`(send` *a-comment-box:snip* `make-editor)` → `(is-a?/c text%)`

    Overrides `make-editor` in `editor-snip:decorated%`.

    Makes an instance of

      `(scheme:text-mixin text:keymap%)`

`(send` *a-comment-box:snip* `make-snip)` → `(is-a?/c comment-snip%)`

    Overrides `make-snip` in `editor-snip:decorated%`.

    Returns an instance of the `comment-snip%` class.

`(send` *a-comment-box:snip* `get-corner-bitmap)`
→ `(is-a?/c bitmap%)`

    Overrides `get-corner-bitmap` in `editor-snip:decorated-mixin`.

    Returns the semicolon bitmap from the file

      `(build-path (collection-path "icons") "semicolon.gif")`

`(send` *a-comment-box:snip* `get-position)`
→ `(symbols 'left-top 'top-right)`

    Overrides `get-position` in `editor-snip:decorated-mixin`.

    Returns `'left-top`

`(send` *a-comment-box:snip* `get-text)` → `string`

    Overrides `get-text` in `snip%`.

    Returns the same string as the super method, but with newlines replaced by newline-semicolon-space.

`(send` *a-comment-box:snip* `get-menu)` → `(is-a?/c popup-menu%)`

    Overrides `get-menu` in `editor-snip:decorated-mixin`.

    Returns a menu with a single item to change the box into semicolon comments.

8 Editor Snip

```
editor-snip:decorated<%> : interface?
```
 implements: `editor-snip%`

```
(send an-editor-snip:decorated get-corner-bitmap)
 → (or/c false/c (is-a?/c bitmap%))
```
 Returns a bitmap that is drawn in the upper-right corner of this snip.

```
(send an-editor-snip:decorated get-color)
 → (or/c string? (is-a?/c color%))
```
 Returns the color used to draw the background part of the snip.

```
(send an-editor-snip:decorated get-menu)
 → (or/c false/c (is-a?/c popup-menu%))
```
 Returns a popup menu that is used when clicking on the top part of the snip.

```
(send an-editor-snip:decorated get-position)
 → (symbols 'top-right 'left-top)
```
 Returns the location of the image and the clickable region. The symbol `'top-right` indicates top portion is clickable and icon on right. The symbol `'left-top` means left portion is clickable and icon on top.

```
(send an-editor-snip:decorated reset-min-sizes) → void?
```
 Sets the minimum sizes based on the result of `get-corner-bitmap`.

```
editor-snip:decorated-mixin : (class? . -> . class?)
```
 argument extends/implements: `editor-snip%`
 result implements: `editor-snip:decorated<%>`

```
(send an-editor-snip:decorated get-corner-bitmap)
 → (or/c false/c (is-a?/c bitmap%))
```
 Returns `#f`.

```
(send an-editor-snip:decorated get-color)
 → (or/c string? (is-a?/c color%))
```
 Returns

```
        (if (preferences:get 'framework:white-on-black?)
            "white"
            "black")
```

```
(send an-editor-snip:decorated get-menu)
 → (or/c false/c (is-a?/c popup-menu%))
```
     Returns #f.

```
(send an-editor-snip:decorated get-position)
 → (symbols 'top-right 'left-top)
```
     Returns 'top-right.

```
editor-snip:decorated% : class?
  superclass: (editor-snip:decorated-mixin editor-snip%)
```

```
(new editor-snip:decorated% ...superclass-args...)
 → (is-a?/c editor-snip:decorated%)
```
     Invokes the super constructor with the keyword editor as a call to make-editor.

```
(send an-editor-snip:decorated make-snip)
 → (is-a?/c editor-snip:decorated%)
```
     This method should return an instance of the class it is invoked in. If you create a subclass of this class, be sure to override this method and have it create instances of the subclass.

```
(send an-editor-snip:decorated make-editor)
 → (is-a?/c editor<%>)
```
     Creates an editor to be used in this snip.

```
(send an-editor-snip:decorated copy)
 → (is-a?/c editor-snip:decorated%)
```
     Uses the make-editor and make-snip methods to create a copy of this snip, as follows:

```
#lang (let ([snip (make-snip)]) (send snip set-editor
(send (get-editor) copy-self)) (send snip set-style (get-
style)) snip)
```

```
editor-snip:decorated-snipclass% : class?
  superclass: snip-class%
```

23

```
(send an-editor-snip:decorated-snipclass make-snip stream-in)
 → -editor-snip:decorated<%>
  stream-in : (is-a?/c editor-stream-in%)
```

Returns an instance of editor-snip:decorated%.

```
(send an-editor-snip:decorated-snipclass read stream-in)
 → editor-snip:decorated<%>
  stream-in : (is-a?/c editor-stream-in%)
```

Calls make-snip to get an object and then invokes its editor<%>'s read-from-file method in order to read a snip from stream-in, eg:

```
(let ([snip (make-snip stream-in)])
  (send (send snip get-editor) read-from-file stream-in #f)
  snip)
```

9   Editor

`editor:basic<%>` : `interface?`
  implements: `editor<%>`

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

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

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

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

> 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* `run-after-edit-sequence` *thunk*
                                                 [*tag*]) → `void`

  thunk : `(-> void)`
  tag : `(union symbol? |#f|)` = `|#f|`

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

(send *an-editor:basic* `get-top-level-window`)
 → `(union |#f| (is-a?/c top-level-window<%>))`

> Returns the `top-level-window<%>` currently associated with this buffer.
>
> This does not work for embedded editors.

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

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

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

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 load-file/gui-error [filename
                                           format
                                           show-errors?])
```
→ boolean?
  filename : (union string |#f|) = |#f|
  format : (union 'guess 'standard 'text 'text-force-cr 'same 'copy)
          = 'guess
  show-errors? : boolean = |#t|

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 on-close) → void
```

This method is called when an editor is closed. Typically, this method is called
when the frame containing the editor is closed, but in some cases an editor
is considered "closed" before the frame it is in is closed (e.g., when a tab in
DrRacket is closed), and thus on-close will be called at that point.

See also can-close? and close.

Does nothing.

```
(send an-editor:basic can-close?) → boolean?
```

This method is called to query the editor if is okay to close the editor. Although
there is no visible effect associated with closing an editor, there may be some
cleanup actions that need to be run when the user is finished with the editor
(asking if it should be saved, for example).

See also on-close and close.

Returns #t.

```
(send an-editor:basic close) → boolean?
```

This method is merely

```
(if (can-close?)
    (begin (on-close) #t)
    #f)
```

It is intended as a shorthand, helper method for closing an editor. See also `can-close?` and `on-close`.

```
(send an-editor:basic get-filename/untitled-name) → string
```

Returns the printed version of the filename for this editor. If the editor doesn't yet have a filename, it returns a symbolic name (something like "Untitled").

```
(send an-editor:basic get-pos/text event)
 → (or/c false/c (is-a?/c editor<%>))
   (or/c false/c number?)
  event : (is-a?/c mouse-event%)
```

This method's first result is `#f` when the mouse event does not correspond to a location in the editor.

If the first result is an `text%` object, then the second result will be a position in the editor and otherwise the second result will be `#f`.

The `editor<%>` object will always be the nearest enclosing editor containing the mouse click.

```
editor:basic-mixin : (class? . -> . class?)
```
  argument extends/implements: `editor<%>`
  result implements: `editor:basic<%>`

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.

```
(send an-editor:basic can-save-file? filename
                                    format) → boolean?
```

filename : `string`
format : `symbol?`

>Augments `can-save-file?` in `editor<%>`.

>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.

(`send` *an-editor:basic* `after-save-file` *success?*) → `void`
  success? : `boolean?`

>Augments `after-save-file` in `editor<%>`.

>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?`.

(`send` *an-editor:basic* `after-load-file` *success?*) → `void`
  success? : `boolean?`

>Augments `after-load-file` in `editor<%>`.

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

(`send` *an-editor:basic* `on-focus` *on?*) → `void`
  on? : `boolean?`

>Overrides `on-focus` in `editor<%>`.

>Manages the state to implement `has-focus?`

(`send` *an-editor:basic* `on-edit-sequence`) → `boolean?`

>Augments `on-edit-sequence` in `editor<%>`.

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

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

>Augments `after-edit-sequence` in `editor<%>`.

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

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

>Overrides `on-new-box` in `editor<%>`.

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

```
(send an-editor:basic on-new-image-snip filname
                                        kind
                                        relative-path?
                                        inline?)
 → (is-a?/c image-snip%)
  filname : (or/c path? false/c)
  kind : (one-of/c 'unknown 'gif 'jpeg 'xbm 'xpm 'bmp 'pict)
  relative-path? : any/c
  inline? : any/c
```

Overrides `on-new-image-snip` in `editor<%>`.

```
    (super on-new-image-snip
           (if (eq? kind 'unknown) 'unknown/mask kind)
           relative-path?
           inline?)
```

```
(send an-editor:basic get-file directory) → string
  directory : (or/c path-string? false/c)
```

Overrides `get-file` in `editor<%>`.

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

```
(send an-editor:basic put-file directory
                               default-name) → string
  directory : (or/c path? false/c)
  default-name : (or/c path? false/c)
```

Overrides `put-file` in `editor<%>`.

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

```
editor:standard-style-list<%> : interface?
  implements: editor<%>
```

This interface is implemented by the results of `editor:standard-style-list-mixin`.

```
editor:standard-style-list-mixin : (class? . -> . class?)
```

argument extends/implements: `editor<%>`
result implements: `editor:standard-style-list<%>`

The mixin adds code to the initialization of the class that sets the editor's style list (via `set-style-list`) to the result of `editor:get-standard-style-list`.

In addition, it calls `set-load-overwrites-styles` with `#f`. This ensures that saved files with different settings for the style list do not clobber the shared style list.

`editor:keymap<%>` : `interface?`
  implements: `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`.

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

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

 The result of this method should not change – that is, it should return the same list of keymaps each time it is called.

 See also `editor:add-after-user-keymap`.

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

`editor:keymap-mixin` : `(class? . -> . class?)`
  argument extends/implements: `editor:basic<%>`
  result implements: `editor:keymap<%>`

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

`editor:autowrap<%>` : `interface?`
  implements: `editor:basic<%>`

Classes implementing this interface keep the `auto-wrap` state set based on the `'framework:auto-set-wrap?` preference (see `preferences:get` for more information 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`.

`editor:autowrap-mixin` : `(class? . -> . class?)`
  argument extends/implements: `editor:basic<%>`
  result implements: `editor:autowrap<%>`

See `editor:autowrap<%>`

`editor:file<%>` : `interface?`
  implements: `editor:keymap<%>`

Objects supporting this interface are expected to support files.

`(send an-editor:file get-can-close-parent)`
`→ (union false (is-a/c? frame%) (is-a/c? dialog%))`

> The result of this method is used as the parent for the dialog that asks about closing.
>
> Defaultly returns `#f`.

`(send an-editor:file update-frame-filename)` `→ void`

> Attempts to find a frame that displays this editor. If it does, it updates the frame's title based on a new filename in the editor.

`(send an-editor:file allow-close-with-no-filename?)` `→ boolean?`

> This method indicates if closing the file when it hasn't been saved is a reason to alert the user. See also `can-close?`.
>
> Defaultly returns `#f`.

`(send an-editor:file user-saves-or-not-modified? allow-cancel?)`
`→ boolean?`

allow-cancel? : #t

> If the file has not been saved, this prompts the user about saving and, if the user says to save, then it saves the file.

> The result is #t if the save file is up to date, or if the user says it is okay to continue without saving. Generally used when closing the file or quiting the app.

`editor:file-mixin` : `(class? . -> . class?)`
  argument extends/implements: `editor:keymap<%>`
  result 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.

`(send` *an-editor:file* `set-filename` *name*
                                        [*temp?*]`)` → `void`
  name : `string`
  temp? : `boolean` = |#f|

> Overrides `set-filename` in `editor<%>`.

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

`(send` *an-editor:file* `can-close?)` → `boolean?`

> Augments `can-close?` in `editor:basic<%>`.

> If the `allow-close-with-no-filename?` method returns #f, this method checks to see if the file has been saved at all yet. If not, it asks the user about saving (and saves if they ask).

> If the `allow-close-with-no-filename?` method returns #t, this method does as before, except only asks if the editor's `get-filename` method returns a path.

> Also calls inner.

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

> Overrides `get-keymaps` in `editor:keymap<%>`.

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

```
editor:backup-autosave<%> : interface?
  implements: editor:basic<%>
```

Classes matching this interface support backup files and autosaving.

```
(send an-editor:backup-autosave backup?) → boolean?
```
> Indicates weather this `editor<%>` should be backed up.
>
> Returns the value of the `preferences:get` applied to `'framework:backup-files?`.

```
(send an-editor:backup-autosave autosave?) → boolean?
```
> Indicates weather this `editor<%>` should be autosaved.
>
> Returns `#t`.

```
(send an-editor:backup-autosave do-autosave)
 → (union |#f| string)
```
> This method is called to perform the autosaving. See also `autosave:register`
>
> 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<%>`.
>
> Returns the filename where the autosave took place, or `#f` if none did.

```
(send an-editor:backup-autosave remove-autosave) → void
```
> This method removes the autosave file associated with this `editor<%>`.

```
editor:backup-autosave-mixin : (class? . -> . class?)
  argument extends/implements: editor:basic<%>
  result implements: editor:backup-autosave<%>
                     autosave:autosavable<%>
```

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.

```
(send an-editor:backup-autosave on-save-file filename
                                             format)  → bool
  filename : path?
  format : (one-of/c 'guess  'standard  'text  'text-force-cr  'same  'copy)
```

33

Augments `on-save-file` in `editor<%>`.

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`

`(send` *an-editor:backup-autosave* `on-close)` → `void`

Augments `on-close` in `editor:basic<%>`.

Deletes the autosave file and turns off autosaving.

`(send` *an-editor:backup-autosave* `on-change)` → `void`

Augments `on-change` in `editor<%>`.

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

`(send` *an-editor:backup-autosave* `set-modified` *modified?*`)` → `void`
  modified? : `any/c`

Overrides `set-modified` in `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.

`editor:info<%>` : `interface?`
  implements: `editor:basic<%>`

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

`editor:info-mixin` : `(class? . -> . class?)`
  argument extends/implements: `editor:basic<%>`
  result implements: `editor:info<%>`

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

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

Overrides `lock` in `editor<%>`.

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

```
(editor:set-default-font-color color) → void?
  color : (is-a?/c color%)
```

Sets the color of the style named `editor:get-default-color-style-name`.

```
(editor:get-default-color-style-name) → string?
```

The name of the style (in the list returned by `editor:get-standard-style-list`) that holds the default color.

```
(editor:set-standard-style-list-delta name
                                       delta) → void?
  name : string?
  delta : (is-a?/c style-delta%)
```

Finds (or creates) the style named by `name` in the result of `editor:get-standard-style-list` and sets its delta to `delta`.

If the style named by `name` is already in the style list, it must be a delta style.

```
(editor:set-standard-style-list-pref-callbacks) → any
```

Installs the font preference callbacks that update the style list returned by `editor:get-standard-style-list` based on the font preference symbols.

```
(editor:get-standard-style-list) → (is-a?/c style-list%)
```

Returns a style list that is used for all instances of `editor:standard-style-list%`.

```
(editor:add-after-user-keymap keymap
                              keymaps)
 → (listof (is-a?/c keymap%))
  keymap : (is-a?/c keymap%)
  keymaps : (listof (is-a?/c keymap%))
```

Returns a list that contains all of the keymaps in `keymaps`, in the same relative order, but also with `keymap`, where `keymap` is now the first keymap after `keymap:get-user` (if that keymap is in the list.)

10 Exit

`(exit:exiting?)` → `boolean?`

Returns `#t` to indicate that an exit operation is taking place. Does not indicate that the app will actually exit, since the user may cancel the exit.

See also `exit:insert-on-callback` and `exit:insert-can?-callback`.

`(exit:set-exiting` *exiting?*`)` → `void?`
  exiting? : `boolean?`

Sets a flag that affects the result of `exit:exiting?`.

`(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: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:can-exit?)` → `boolean?`

Calls the "can-callbacks" and returns their results. See `exit:insert-can?-callback` for more information.

`(exit:on-exit)` → `void?`

Calls the "on-callbacks". See `exit:insert-on-callback` for more information.

`(exit:exit)` → `any`

`exit:exit` performs four actions:

- sets the result of the `exit:exiting?` function to `#t`.

- invokes the exit-callbacks, with `exit:can-exit?` if none of the "can?" callbacks return `#f`,

- invokes `exit:on-exit` and then

- queues a callback that calls `exit` (a mzscheme procedure) and (if `exit` returns) sets the result of `exit:exiting?` back to `#t`.

`(exit:user-oks-exit)` → `boolean?`

Opens a dialog that queries the user about exiting. Returns the user's decision.

11 Finder

```
(finder:dialog-parent-parameter)
 → (or/c false/c (is-a?/c dialog%) (is-a?/c frame%))
(finder:dialog-parent-parameter parent) → void?
  parent : (or/c false/c (is-a?/c dialog%) (is-a?/c frame%))
```

This parameter 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: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) → (listof (list/c string? string?))
(finder:default-filters filters) → void?
  filters : (listof (list/c string? string?))
```

This parameter controls the default filters 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:common-put-file [name
                          directory
                          replace?
                          prompt
                          filter
                          filter-msg
                          parent])    → (or/c false/c path?)
  name : string? = "Untitled"
  directory : (or/c false/c path?) = #f
  replace? : boolean? = #f
  prompt : string? = "Select File"
  filter : (or/c false/c byte-regexp?) = #f
```

```
filter-msg :  string?
            = "That filename does not have the right form."
parent : (or/c (is-a?/c top-level-window<%>) false/c)
       = (finder:dialog-parent-parameter)
```

This procedure queries the user for a single filename, using a platform-independent dialog box. Consider using `finder:put-file` instead of this function.

```
(finder:common-get-file [directory
                         prompt
                         filter
                         filter-msg
                         parent])    → (or/c path? false/c)
  directory : (or/c path? false/c) = #f
  prompt : string? = "Select File"
  filter : (or/c byte-regexp? false/c) = #f
  filter-msg :  string?
             = "That filename does not have the right form."
  parent : (or/c false/c (is-a?/c top-level-window<%>)) = #f
```

This procedure queries the user for a single filename, using a platform-independent dialog box. Consider using `finder:get-file` instead of this function.

```
(finder:std-put-file [name
                      directory
                      replace?
                      prompt
                      filter
                      filter-msg
                      parent])    → (or/c false/c path?)
  name : string? = "Untitled"
  directory : (or/c false/c path?) = #f
  replace? : boolean? = #f
  prompt : string? = "Select File"
  filter : (or/c false/c byte-regexp?) = #f
  filter-msg :  string?
             = "That filename does not have the right form."
  parent : (or/c (is-a?/c top-level-window<%>) false/c)
         = (finder:dialog-parent-parameter)
```

This procedure queries the user for a single filename, using a platform-dependent dialog box. Consider using `finder:put-file` instead of this function.

```
(finder:std-get-file [directory
                      prompt
                      filter
                      filter-msg
                      parent])    → (or/c path? false/c)
  directory : (or/c path? false/c) = #f
  prompt : string? = "Select File"
  filter : (or/c byte-regexp? false/c) = #f
  filter-msg :  string?
             = "That filename does not have the right form."
  parent : (or/c false/c (is-a?/c top-level-window<%>)) = #f
```

This procedure queries the user for a single filename, using a platform-dependent dialog box. Consider using `finder:get-file` instead of this function.

```
(finder:put-file [name
                  directory
                  replace?
                  prompt
                  filter
                  filter-msg
                  parent])    → (or/c false/c path?)
  name : string? = "Untitled"
  directory : (or/c false/c path?) = #f
  replace? : boolean? = #f
  prompt : string? = "Select File"
  filter : (or/c false/c byte-regexp?) = #f
  filter-msg :  string?
             = "That filename does not have the right form."
  parent : (or/c (is-a?/c top-level-window<%>) false/c)
         = (finder:dialog-parent-parameter)
```

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:get-file [directory
                  prompt
                  filter
                  filter-msg
                  parent])    → (or/c path? false/c)
  directory : (or/c path? false/c) = #f
  prompt : string? = "Select File"
```

```
  filter : (or/c byte-regexp? string? false/c) = #f
  filter-msg :  string?
              = "That filename does not have the right form."
  parent : (or/c false/c (is-a?/c top-level-window<%>)) = #f
```

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:common-get-file-list [directory
                              prompt
                              filter
                              filter-msg
                              parent])
 → (or/c (listof path?) false/c)
  directory : (or/c false/c path?) = #f
  prompt : string? = "Select File"
  filter : (or/c false/c byte-regexp?) = #f
  filter-msg :  string?
              = "That filename does not have the right form."
  parent : (or/c false/c (is-a?/c top-level-window<%>)) = #f
```

This procedure queries the user for a list of filenames, using a platform-independent dialog box.

12 Frame

```
frame:basic<%> : interface?
   implements: frame%
```

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

```
(send a-frame:basic get-area-container%)
→ (is-a?/c 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)
→ (instance (is-a?/c area-container<%>))
```

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

```
(send a-frame:basic get-menu-bar%) → (subclass?/c menu-bar%)
```

> The result of this method is used to create the initial menu bar for this frame.
>
> Return `menu-bar%`.

```
(send a-frame:basic make-root-area-container class
                                            parent)
→ (instance (is-a?/c area-container<%>))
 class : (is-a?/c area-container<%>)
 parent : (instance (is-a?/c 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.

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

(send *a-frame:basic* close) → void

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* editing-this-file? *filename*) → boolean?
  filename : path

Indicates if this frame contains this buffer (and can edit that file).

Returns #f.

(send *a-frame:basic* get-filename [*temp*]) → (or/c #f path)
  temp : (or/c #f (box boolean?)) = #f

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

Defaultly returns #f.

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

(send *a-frame:basic* make-visible *filename*) → void
  filename : string

Makes the file named by *filename* visible (intended for use with tabbed editing).

---

frame:basic-mixin : (class? . -> . class?)

argument extends/implements: `frame%`
result 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%` or `control<%>` objects this frame as parent. Instead, use the result of the `get-area-container` method.

This mixin also creates a menu bar for the frame, as the frame is initialized. It uses the class returned by `get-menu-bar%`. It only passes the frame as an initialization argument. In addition, it creates the windows menu in the menu bar.

This mixin calls its `accept-drop-files` with `#t`.

It also calls its `set-icon` method according to the current value of `frame:current-icon`.

See also `frame:reorder-menus`.

(send `a-frame:basic` show `on?`) → void
  `on?` : `boolean?`

> Overrides `show` in `top-level-window<%>`.
>
> Calls the super method.
>
> When `on?` is `#t`, inserts the frame into the frame group and when it is `#f`, removes the frame from the group.

(send `a-frame:basic` can-exit?) → `boolean?`

> Overrides `can-exit?` in `top-level-window<%>`.
>
> This, together with `on-exit` mimics `exit:exit`.
>
> First, it calls `exit:set-exiting` with `#t`. Then, it calls `exit:can-exit?`. If it returns `#t`, so does this method. If it returns `#f`, this method calls `exit:set-exiting` with `#f`.

(send `a-frame:basic` on-exit) → void

> Overrides `on-exit` in `top-level-window<%>`.
>
> Together with `can-exit?` this mimics the behavior of `exit:exit`.
>
> Calls `exit:on-exit` and then queues a callback to call Racket's `exit` function. If that returns, it calls `exit:set-exiting` to reset that flag to `#f`.

(send `a-frame:basic` on-superwindow-show `shown?`) → void
  `shown?` : `any/c`

> Overrides `on-superwindow-show` in `window<%>`.

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

(send `a-frame:basic` `on-drop-file` *pathname*) → `void`
  pathname : `string`

Overrides `on-drop-file` in `window<%>`.

Calls `handler:edit-file` with *pathname* as an argument.

(send `a-frame:basic` `after-new-child`) → `void`

Overrides `after-new-child` in `area-container<%>`.

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

`frame:size-pref<%>` : `interface?`
  implements: `frame:basic<%>`

`frame:size-pref-mixin` : (`class?` . `->` . `class?`)
  argument extends/implements: `frame:basic<%>`
  result implements: `frame:size-pref<%>`

```
(new frame:size-pref-mixin
  [size-preferences-key size-preferences-key]
  [label label]
 [[parent parent]
  [x x]
  [y y]
  [style style]
  [enabled enabled]
  [border border]
  [spacing spacing]
  [alignment alignment]
  [min-width min-width]
  [min-height min-height]
  [stretchable-width stretchable-width]
  [stretchable-height stretchable-height]])
→ (is-a?/c frame:size-pref-mixin)
  size-preferences-key : symbol?
  label : label-string?
```

45

```
parent : (or/c (is-a?/c frame%) false/c) = #f
x : (or/c (integer-in -10000 10000) false/c) = #f
y : (or/c (integer-in -10000 10000) false/c) = #f
style : (listof (or/c 'no-resize-border 'no-caption 'no-system-menu 'hide-menu-bar 'mdi
       = null
enabled : any/c = #t
border : (integer-in 0 1000) = 0
spacing : (integer-in 0 1000) = 0
alignment : (list/c (or/c 'left 'center 'right) (or/c 'top 'center 'bottom))
           = '(center top)
min-width : (integer-in 0 10000) = graphical-minimum-width
min-height : (integer-in 0 10000) = graphical-minimum-height
stretchable-width : any/c = #t
stretchable-height : any/c = #t
```

The size `size-preferences-key` symbol is used with `preferences:get`
and `preferences:set` to track the current size.

Passes the `width` and `height` initialization arguments to the superclass based
on the current value of the preference.

See also `frame:setup-size-pref`.

```
(send a-frame:size-pref on-size width
                                 height) → void
  width : number
  height : number
```

Overrides `on-size` in `window<%>`.

Updates the preferences, according to the width and height. The preferences
key is the one passed to the initialization argument of the class.

`frame:register-group<%>` : `interface?`

Frames that implement this interface are registered with the group. See `group:get-the-frame-group` and `frame:register-group-mixin`.

```
frame:register-group-mixin : (class? . -> . class?)
  argument extends/implements: frame:basic<%>
  result implements: frame:register-group<%>
```

During initialization, calls `insert-frame`with `this`.

(send *a-frame:register-group* `can-close?`) → `bool`

> Augments `can-close?` in `top-level-window<%>`.
>
> Calls the inner method, with a default of `#t`. If that returns `#t`, it checks for one of the these three conditions:
>
> - `exit:exiting?` returns `#t`
> - there is more than one frame in the group returned by `group:get-the-frame-group`, or
> - the procedure `exit:user-oks-exit` returns `#t`.
>
> If any of those conditions hold, the method returns `#t`.

(send *a-frame:register-group* `on-close`) → `void`

> Augments `on-close` in `top-level-window<%>`.
>
> First calls the inner method. Next, calls the `remove-frame` method of the result of `group:get-the-frame-group` with `this` as an argument. Finally, unless `exit:exiting?` returns `#t`, and if there are no more frames open, it calls `exit:exit`.

(send *a-frame:register-group* `on-activate` *on?*) → `void`
  on? : `boolean?`

> Overrides `on-activate` in `top-level-window<%>`.
>
> Calls `set-active-frame` with `this` when *on?* is true.

`frame:status-line<%>` : `interface?`
  implements: `frame:basic<%>`

The mixin that implements this interface provides an interface to a set of status lines at the bottom of this frame.

Each status line must be opened with `open-status-line` before any messages are shown in the status line and once `close-status-line` is called, no more messages may be displayed, unless the status line is re-opened.

The screen space for status lines is not created until `update-status-line` is called with a string. Additionally, the screen space for one status line is re-used when by another status line when the first passes `#f` to `update-status-line`. In this manner, the status line frame avoids opening too many status lines and avoids flashing the status lines open and closed too often.

```
(send a-frame:status-line open-status-line id) → void
  id : symbol?
```
> Creates a new status line identified by the symbol argument. The line will not appear in the frame until a message is put into it, via `update-status-line`.

```
(send a-frame:status-line close-status-line id) → void
  id : symbol?
```
> Closes the status line `id`.

```
(send a-frame:status-line update-status-line id
                                              status) → void
  id : symbol?
  status : (or/c #f string)
```
> Updates the status line named by `id` with `status`. If `status` is `#f`, the status line is becomes blank (and may be used by other ids).

```
frame:status-line-mixin : (class? . -> . class?)
  argument extends/implements: frame:basic<%>
  result implements: frame:status-line<%>
```

```
(send a-frame:status-line make-root-area-container class
                                                    parent)
 → (is-a?/c panel%)
  class : (subclass?/c panel%)
  parent : (instanceof (subclass?/c panel%))
```
> Overrides `make-root-area-container` in `frame:basic<%>`.

> Adds a panel at the bottom of the frame to hold the status lines.

```
frame:info<%> : interface?
  implements: frame:basic<%>
```

Frames matching this interface support a status line.

The preference `'framework:show-status-line` controls the visibility of the status line. If it is `#t`, the status line is visible and if it is `#f`, the status line is not visible (see `preferences:get` for more info about preferences)

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

> 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 lock-status-changed) → void
```

> This method is called when the lock status of the editor<%> changes.
>
> Updates the lock icon in the status line panel.

```
(send a-frame:info update-info) → void
```

> This method updates all of the information in the panel.

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

> 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 get-info-canvas) → (instance canvas:basic%)
```

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

```
(send a-frame:info get-info-editor)
 → (or/c #f (is-a?/c editor<%>))
```

> Override this method to specify the editor that the status line contains information about.
>
> Returns the result of get-editor.

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

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

```
(send a-frame:info show-info) → void
```

> Shows the info panel.

See also `is-info-hidden?`.

`(send` *a-frame:info* `hide-info)` → `void`

Hides the info panel.

See also `is-info-hidden?`.

`(send` *a-frame:info* `is-info-hidden?)` → `boolean?`

Result indicates if the show info panel has been explicitly hidden with `hide-info`.

If this method returns `#t` and `(preferences:get 'framework:show-status-line)` is `#f`, then the info panel will not be visible. Otherwise, it is visible.

`frame:info-mixin` : `(class? . -> . class?)`
  argument extends/implements: `frame:basic<%>`
  result implements: `frame:info<%>`

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

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

`(send` *a-frame:info* `make-root-area-container` *class*
                                                parent)`
 → `(instance` `area-container<%>)`
  class : `(subclass?/c` `area-container<%>)`
  parent : `(is-a?/c` `area-container<%>)`

Overrides `make-root-area-container` in `frame:basic<%>`.

Builds an extra panel for displaying various information.

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

Augments `on-close` in `top-level-window<%>`.

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

`frame:text-info<%>` : `interface?`
  implements: `frame:info<%>`

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

`(send ` *a-frame:text-info* ` set-macro-recording` *on?*`)` $\rightarrow$ `void`
  on? : `boolean?`

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

`(send ` *a-frame:text-info* ` overwrite-status-changed)` $\rightarrow$ `void`

> 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* ` anchor-status-changed)` $\rightarrow$ `void`

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

`(send ` *a-frame:text-info* ` editor-position-changed)` $\rightarrow$ `void`

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

`frame:text-info-mixin` : `(class? . -> . class?)`
  argument extends/implements: `frame:info<%>`
  result implements: `frame:text-info<%>`

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

`(send ` *a-frame:text-info* ` on-close)` $\rightarrow$ `void`

> Augments `on-close` in `top-level-window<%>`.

> removes a preferences callback for `'framework:line-offsets`. See `preferences:add-callback` for more information.

`(send ` *a-frame:text-info* ` update-info)` $\rightarrow$ `void`

> Overrides `update-info` in `frame:info<%>`.

> Calls `overwrite-status-changed`, `anchor-status-changed`, and `editor-position-changed`.

`frame:pasteboard-info<%>` : `interface?`
  implements: `frame:info<%>`

`frame:pasteboard-info-mixin` : `(class? . -> . class?)`
  argument extends/implements: `frame:basic<%>`
  result implements: `frame:pasteboard-info<%>`

`frame:standard-menus<%>` : `interface?`
  implements: `frame:basic<%>`

`(send a-frame:standard-menus on-close)` → `void?`

      Removes the preferences callbacks for the menu items

`(send a-frame:standard-menus get-menu%)`
 → `(is-a?/c menu:can-restore-underscore-menu%)`

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

`(send a-frame:standard-menus get-menu-item%)`
 → `(is-a?/c menu:can-restore-menu-item%)`

      The result of this method is used as the class for creating the menu items in this frame.

      Defaultly returns `menu:can-restore-menu-item`.

`(send a-frame:standard-menus get-checkable-menu-item%)`
 → `(is-a?/c menu:can-restore-checkable-menu-item%)`

      The result of this method is used as the class for creating checkable menu items in this class.

      Defaultly returns `menu:can-restore-checkable-menu-item`.

`(send a-frame:standard-menus get-file-menu)` → `(is-a?/c menu%)`

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

`(send a-frame:standard-menus get-edit-menu)` → `(is-a?/c menu%)`

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

`(send a-frame:standard-menus get-help-menu)` → `(is-a?/c menu%)`

      Returns the help menu. See also `get-menu%`.

```
(send a-frame:standard-menus file-menu:get-new-item)
 → (or/c false/c (is-a?/c menu-item%))
```

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `file-menu:create-new?`).

```
(send a-frame:standard-menus file-menu:create-new?) → boolean?
```

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

Defaults to `#t`.

```
(send a-frame:standard-menus file-menu:new-callback item
                                                     control)
 → void?
  item : (is-a?/c menu-item%)
  control : (is-a?/c control-event%)
```

Defaults to

```
    (begin (handler:edit-file #f) #t)
```

```
(send a-frame:standard-menus file-menu:new-on-demand menu-item)
 → void?
  menu-item : (is-a?/c menu-item%)
```

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

Defaults to

```
    (void)
```

```
(send a-frame:standard-menus file-menu:new-string) → string?
```

The result of this method is used as the name of the `menu-item%`.

Defaults to `(string-constant new-menu-item)`.

```
(send a-frame:standard-menus file-menu:new-help-string)
 → string?
```

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

Defaults to `(string-constant new-info)`.

```
(send a-frame:standard-menus file-menu:between-new-and-open menu)
 → void?
  menu : (is-a?/c menu-item%)
```

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

```
(send a-frame:standard-menus file-menu:get-open-item)
 → (or/c false/c (is-a?/c menu-item%))
```

> This method returns the menu-item% object corresponding to this menu item,
> if it has been created (as controlled by file-menu:create-open?).

```
(send a-frame:standard-menus file-menu:create-open?)
 → boolean?
```

> The result of this method determines if the corresponding menu item is created.
> Override it to control the creation of the menu item.
>
> Defaults to #t.

```
(send a-frame:standard-menus file-menu:open-callback item
                                                     control)
 → void?
  item : (is-a?/c menu-item%)
  control : (is-a?/c control-event%)
```

> Defaults to
>
> (begin (handler:open-file) #t)

```
(send a-frame:standard-menus file-menu:open-on-demand menu-item)
 → void?
  menu-item : (is-a?/c menu-item%)
```

> The menu item's on-demand proc calls this method.
>
> Defaults to
>
> (void)

```
(send a-frame:standard-menus file-menu:open-string) → string?
```

> The result of this method is used as the name of the menu-item%.
>
> Defaults to (string-constant open-menu-item).

```
(send a-frame:standard-menus file-menu:open-help-string)
 → string?
```

> The result of this method is used as the help string when the menu-item% object
> is created.
>
> Defaults to (string-constant open-info).

```
(send a-frame:standard-menus file-menu:get-open-recent-item)
 → (or/c false/c (is-a?/c menu-item%))
```

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `file-menu:create-open-recent?`).

`(send a-frame:standard-menus file-menu:create-open-recent?)`
`→ boolean?`

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

Defaults to `#t`.

`(send a-frame:standard-menus file-menu:open-recent-callback x`
`                                                              y)`
`→ void?`
`  x : (is-a?/c menu-item%)`
`  y : (is-a?/c control-event%)`

Defaults to

`(void)`

`(send a-frame:standard-menus file-menu:open-recent-on-demand menu)`
`→ void?`
`  menu : (is-a?/c menu-item%)`

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

Defaults to

`(handler:install-recent-items menu)`

`(send a-frame:standard-menus file-menu:open-recent-string)`
`→ string?`

The result of this method is used as the name of the `menu-item%`.

Defaults to `(string-constant open-recent-menu-item)`.

`(send a-frame:standard-menus file-menu:open-recent-help-string)`
`→ string?`

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

Defaults to `(string-constant open-recent-info)`.

`(send a-frame:standard-menus file-menu:between-open-and-revert menu)`
`→ void?`
`  menu : (is-a?/c menu-item%)`

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

```
(send a-frame:standard-menus file-menu:get-revert-item)
→ (or/c false/c (is-a?/c menu-item%))
```

> This method returns the menu-item% object corresponding to this menu item,
> if it has been created (as controlled by file-menu:create-revert?).

```
(send a-frame:standard-menus file-menu:create-revert?)
→ boolean?
```

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

> Defaults to #f.

```
(send a-frame:standard-menus file-menu:revert-callback
  item
  control)
→ void?
  item : (is-a?/c menu-item%)
  control : (is-a?/c control-event%)
```

> Defaults to

> > (void)

```
(send a-frame:standard-menus file-menu:revert-on-demand menu-item)
→ void?
  menu-item : (is-a?/c menu-item%)
```

> The menu item's on-demand proc calls this method.

> Defaults to

> > (void)

```
(send a-frame:standard-menus file-menu:revert-string)
→ string?
```

> The result of this method is used as the name of the menu-item%.

> Defaults to (string-constant revert-menu-item).

```
(send a-frame:standard-menus file-menu:revert-help-string)
→ string?
```

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

> Defaults to (string-constant revert-info).

```
(send a-frame:standard-menus file-menu:between-revert-and-save menu)
→ void?
```

menu : `(is-a?/c menu-item%)`

> This method is called between the addition of the `revert` and the `save` menu-item. Override it to add additional menu items at that point.

`(send` *a-frame:standard-menus* `file-menu:get-save-item)`
→ `(or/c false/c (is-a?/c menu-item%))`

> This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `file-menu:create-save?`).

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

> The result of this method determines if the corresponding menu item is created. Override it to control the creation of the menu item.
>
> Defaults to `#f`.

`(send` *a-frame:standard-menus* `file-menu:save-callback` *item*
                                                              control`)`
→ `void?`
  item : `(is-a?/c menu-item%)`
  control : `(is-a?/c control-event%)`

> Defaults to
>
> > `(void)`

`(send` *a-frame:standard-menus* `file-menu:save-on-demand` *menu-item*`)`
→ `void?`
  menu-item : `(is-a?/c menu-item%)`

> The menu item's on-demand proc calls this method.
>
> Defaults to
>
> > `(void)`

`(send` *a-frame:standard-menus* `file-menu:save-string)` → `string?`

> The result of this method is used as the name of the `menu-item%`.
>
> Defaults to `(string-constant save-menu-item)`.

`(send` *a-frame:standard-menus* `file-menu:save-help-string)`
→ `string?`

> The result of this method is used as the help string when the `menu-item%` object is created.
>
> Defaults to `(string-constant save-info)`.

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

→ `(or/c false/c (is-a?/c menu-item%))`

> This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `file-menu:create-save-as?`).

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

> The result of this method determines if the corresponding menu item is created. Override it to control the creation of the menu item.
>
> Defaults to `#f`.

`(send a-frame:standard-menus file-menu:save-as-callback`
`  item`
`  control)`
→ `void?`
`  item : (is-a?/c menu-item%)`
`  control : (is-a?/c control-event%)`

> Defaults to
>
> `(void)`

`(send a-frame:standard-menus file-menu:save-as-on-demand menu-item)`
→ `void?`
`  menu-item : (is-a?/c menu-item%)`

> The menu item's on-demand proc calls this method.
>
> Defaults to
>
> `(void)`

`(send a-frame:standard-menus file-menu:save-as-string)`
→ `string?`

> The result of this method is used as the name of the `menu-item%`.
>
> Defaults to `(string-constant save-as-menu-item)`.

`(send a-frame:standard-menus file-menu:save-as-help-string)`
→ `string?`

> The result of this method is used as the help string when the `menu-item%` object is created.
>
> Defaults to `(string-constant save-as-info)`.

`(send a-frame:standard-menus file-menu:between-save-as-and-print menu)`
→ `void?`
`  menu : (is-a?/c menu-item%)`

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

```
(send a-frame:standard-menus file-menu:get-print-item)
 → (or/c false/c (is-a?/c menu-item%))
```

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `file-menu:create-print?`).

```
(send a-frame:standard-menus file-menu:create-print?)
 → boolean?
```

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

Defaults to `#f`.

```
(send a-frame:standard-menus file-menu:print-callback item
                                                        control)
 → void?
  item : (is-a?/c menu-item%)
  control : (is-a?/c control-event%)
```

Defaults to

```
(void)
```

```
(send a-frame:standard-menus file-menu:print-on-demand menu-item)
 → void?
  menu-item : (is-a?/c menu-item%)
```

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

Defaults to

```
(void)
```

```
(send a-frame:standard-menus file-menu:print-string) → string?
```

The result of this method is used as the name of the `menu-item%`.

Defaults to `(string-constant print-menu-item)`.

```
(send a-frame:standard-menus file-menu:print-help-string)
 → string?
```

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

Defaults to `(string-constant print-info)`.

```
(send a-frame:standard-menus file-menu:between-print-and-close menu)
 → void?
```

menu : `(is-a?/c menu-item%)`

> This method is called between the addition of the `print` and the `close` menu-item. Override it to add additional menu items at that point.

`(send` *a-frame:standard-menus* `file-menu:get-close-item)`
`→ (or/c false/c (is-a?/c menu-item%))`

> This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `file-menu:create-close?`).

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

> The result of this method determines if the corresponding menu item is created. Override it to control the creation of the menu item.
>
> Defaults to `#t`.

`(send` *a-frame:standard-menus* `file-menu:close-callback` *item*
                                                          control`)`
`→ void?`
  item : `(is-a?/c menu-item%)`
  control : `(is-a?/c control-event%)`

> Defaults to
>
> `(begin (when (can-close?) (on-close) (show #f)) #t)`

`(send` *a-frame:standard-menus* `file-menu:close-on-demand` *menu-item*`)`
`→ void?`
  menu-item : `(is-a?/c menu-item%)`

> The menu item's on-demand proc calls this method.
>
> Defaults to
>
> `(void)`

`(send` *a-frame:standard-menus* `file-menu:close-string)` `→ string?`

> The result of this method is used as the name of the `menu-item%`.
>
> Defaults to `(string-constant close-menu-item)`.

`(send` *a-frame:standard-menus* `file-menu:close-help-string)`
`→ string?`

> The result of this method is used as the help string when the `menu-item%` object is created.
>
> Defaults to `(string-constant close-info)`.

`(send` *a-frame:standard-menus* `file-menu:between-close-and-quit` *menu*`)`

`→ void?`
  `menu : (is-a?/c menu-item%)`

> This method is called between the addition of the `close` and the `quit` menu-item. Override it to add additional menu items at that point.

`(send a-frame:standard-menus file-menu:get-quit-item)`
`→ (or/c false/c (is-a?/c menu-item%))`

> This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `file-menu:create-quit?`).

`(send a-frame:standard-menus file-menu:create-quit?)`
`→ boolean?`

> The result of this method determines if the corresponding menu item is created. Override it to control the creation of the menu item.
>
> Defaults to `(not (current-eventspace-has-standard-menus?))`.

`(send a-frame:standard-menus file-menu:quit-callback item`
                                                      `control)`
`→ void?`
  `item : (is-a?/c menu-item%)`
  `control : (is-a?/c control-event%)`

> Defaults to
>
>   `(when (exit:user-oks-exit) (exit:exit))`

`(send a-frame:standard-menus file-menu:quit-on-demand menu-item)`
`→ void?`
  `menu-item : (is-a?/c menu-item%)`

> The menu item's on-demand proc calls this method.
>
> Defaults to
>
>   `(void)`

`(send a-frame:standard-menus file-menu:quit-string) → string?`

> The result of this method is used as the name of the `menu-item%`.
>
> Defaults to `(if (eq? (system-type) 'windows) (string-constant quit-menu-item-windows) (string-constant quit-menu-item-others))`.

`(send a-frame:standard-menus file-menu:quit-help-string)`
`→ string?`

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

Defaults to `(string-constant quit-info)`.

`(send` *a-frame:standard-menus* `file-menu:after-quit` *menu*`)` → `void?`
　menu : `(is-a?/c menu-item%)`

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

`(send` *a-frame:standard-menus* `edit-menu:get-undo-item)`
→ `(or/c false/c (is-a?/c menu-item%))`

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `edit-menu:create-undo?`).

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

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

Defaults to `#t`.

`(send` *a-frame:standard-menus* `edit-menu:undo-callback` *menu*
　　　　　　　　　　　　　　　　　　　　　　　　evt`)`
→ `void?`
　menu : `(is-a?/c menu-item%)`
　evt : `(is-a?/c control-event%)`

Defaults to

```
(begin (let ((edit (get-edit-target-
object))) (when (and edit (is-a? edit editor<%>)) (send edit do-
edit-operation 'undo)))  #t)
```

`(send` *a-frame:standard-menus* `edit-menu:undo-on-demand` *item*`)`
→ `void?`
　item : `(is-a?/c menu-item%)`

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

Defaults to

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

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

The result of this method is used as the name of the `menu-item%`.

Defaults to `(string-constant undo-menu-item)`.

```
(send a-frame:standard-menus edit-menu:undo-help-string)
 → string?
```

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

Defaults to (`string-constant undo-info`).

```
(send a-frame:standard-menus edit-menu:get-redo-item)
 → (or/c false/c (is-a?/c menu-item%))
```

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `edit-menu:create-redo?`).

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

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

Defaults to `#t`.

```
(send a-frame:standard-menus edit-menu:redo-callback menu
                                                     evt)
 → void?
  menu : (is-a?/c menu-item%)
  evt : (is-a?/c control-event%)
```

Defaults to

```
    (begin (let ((edit (get-edit-target-
object))) (when (and edit (is-a? edit editor<%>)) (send edit do-
edit-operation 'redo)))  #t)
```

```
(send a-frame:standard-menus edit-menu:redo-on-demand item)
 → void?
  item : (is-a?/c menu-item%)
```

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

Defaults to

```
    (let* ((editor (get-edit-target-object)) (enable? (and editor (is-
a? editor editor<%>) (send editor can-do-edit-
operation? 'redo))))  (send item enable enable?))
```

```
(send a-frame:standard-menus edit-menu:redo-string) → string?
```

The result of this method is used as the name of the `menu-item%`.

Defaults to (`string-constant redo-menu-item`).

```
(send a-frame:standard-menus edit-menu:redo-help-string)
→ string?
```

> The result of this method is used as the help string when the `menu-item%` object
> is created.
>
> Defaults to (`string-constant redo-info`).

```
(send a-frame:standard-menus edit-menu:between-redo-and-cut menu)
→ void?
  menu : (is-a?/c menu-item%)
```

> This method is called between the addition of the `redo` and the `cut` menu-item.
> Override it to add additional menu items at that point.

```
(send a-frame:standard-menus edit-menu:get-cut-item)
→ (or/c false/c (is-a?/c menu-item%))
```

> This method returns the `menu-item%` object corresponding to this menu item,
> if it has been created (as controlled by `edit-menu:create-cut?`).

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

> The result of this method determines if the corresponding menu item is created.
> Override it to control the creation of the menu item.
>
> Defaults to `#t`.

```
(send a-frame:standard-menus edit-menu:cut-callback menu
                                                     evt)
→ void?
  menu : (is-a?/c menu-item%)
  evt : (is-a?/c control-event%)
```

> Defaults to
>
> ```
>    (begin (let ((edit (get-edit-target-
> object))) (when (and edit (is-a? edit editor<%>)) (send edit do-
> edit-operation 'cut)))  #t)
> ```

```
(send a-frame:standard-menus edit-menu:cut-on-demand item)
→ void?
  item : (is-a?/c menu-item%)
```

> The menu item's on-demand proc calls this method.
>
> Defaults to
>
> ```
>    (let* ((editor (get-edit-target-object)) (enable? (and editor (is-
> a? editor editor<%>) (send editor can-do-edit-
> operation? 'cut))))  (send item enable enable?))
> ```

`(send a-frame:standard-menus edit-menu:cut-string) → string?`

The result of this method is used as the name of the `menu-item%`.

Defaults to `(string-constant cut-menu-item)`.

`(send a-frame:standard-menus edit-menu:cut-help-string)`
`→ string?`

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

Defaults to `(string-constant cut-info)`.

`(send a-frame:standard-menus edit-menu:between-cut-and-copy menu)`
`→ void?`
  `menu : (is-a?/c menu-item%)`

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

`(send a-frame:standard-menus edit-menu:get-copy-item)`
`→ (or/c false/c (is-a?/c menu-item%))`

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `edit-menu:create-copy?`).

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

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

Defaults to `#t`.

`(send a-frame:standard-menus edit-menu:copy-callback menu`
                                                      `evt)`
`→ void?`
  `menu : (is-a?/c menu-item%)`
  `evt : (is-a?/c control-event%)`

Defaults to

```
(begin (let ((edit (get-edit-target-
object))) (when (and edit (is-a? edit editor<%>)) (send edit do-
edit-operation 'copy)))  #t)
```

`(send a-frame:standard-menus edit-menu:copy-on-demand item)`
`→ void?`
  `item : (is-a?/c menu-item%)`

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

Defaults to

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

`(send a-frame:standard-menus edit-menu:copy-string) → string?`

The result of this method is used as the name of the `menu-item%`.

Defaults to `(string-constant copy-menu-item)`.

`(send a-frame:standard-menus edit-menu:copy-help-string)`
`→ string?`

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

Defaults to `(string-constant copy-info)`.

`(send a-frame:standard-menus edit-menu:between-copy-and-paste menu)`
`→ void?`
  `menu : (is-a?/c menu-item%)`

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

`(send a-frame:standard-menus edit-menu:get-paste-item)`
`→ (or/c false/c (is-a?/c menu-item%))`

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `edit-menu:create-paste?`).

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

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

Defaults to `#t`.

`(send a-frame:standard-menus edit-menu:paste-callback menu`
                                                    `evt)`
`→ void?`
  `menu : (is-a?/c menu-item%)`
  `evt : (is-a?/c control-event%)`

Defaults to

```
(begin (let ((edit (get-edit-target-
object))) (when (and edit (is-a? edit editor<%>)) (send edit do-
edit-operation 'paste)))  #t)
```

66

```
(send a-frame:standard-menus edit-menu:paste-on-demand item)
→ void?
  item : (is-a?/c menu-item%)
```

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

Defaults to

```
(let* ((editor (get-edit-target-object)) (enable? (and editor (is-
a? editor editor<%>) (send editor can-do-edit-
operation? 'paste))))  (send item enable enable?))
```

```
(send a-frame:standard-menus edit-menu:paste-string) → string?
```

The result of this method is used as the name of the `menu-item%`.

Defaults to `(string-constant paste-menu-item)`.

```
(send a-frame:standard-menus edit-menu:paste-help-string)
→ string?
```

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

Defaults to `(string-constant paste-info)`.

```
(send a-frame:standard-menus edit-menu:between-paste-and-clear menu)
→ void?
  menu : (is-a?/c menu-item%)
```

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

```
(send a-frame:standard-menus edit-menu:get-clear-item)
→ (or/c false/c (is-a?/c menu-item%))
```

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `edit-menu:create-clear?`).

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

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

Defaults to `#t`.

```
(send a-frame:standard-menus edit-menu:clear-callback menu
                                                      evt)
→ void?
```

```
menu : (is-a?/c menu-item%)
evt : (is-a?/c control-event%)
```

Defaults to

```
(begin (let ((edit (get-edit-target-
object))) (when (and edit (is-a? edit editor<%>)) (send edit do-
edit-operation 'clear)))  #t)
```

```
(send a-frame:standard-menus edit-menu:clear-on-demand item)
→ void?
  item : (is-a?/c menu-item%)
```

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

Defaults to

```
(let* ((editor (get-edit-target-object)) (enable? (and editor (is-
a? editor editor<%>) (send editor can-do-edit-
operation? 'clear))))  (send item enable enable?))
```

```
(send a-frame:standard-menus edit-menu:clear-string) → string?
```

The result of this method is used as the name of the menu-item%.

Defaults to (if (eq? (system-type) 'windows) (string-constant clear-menu-item-windows) (string-constant clear-menu-item-windows)).

```
(send a-frame:standard-menus edit-menu:clear-help-string)
→ string?
```

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

Defaults to (string-constant clear-info).

```
(send a-frame:standard-menus edit-menu:between-clear-and-select-all menu)
→ void?
  menu : (is-a?/c menu-item%)
```

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

```
(send a-frame:standard-menus edit-menu:get-select-all-item)
→ (or/c false/c (is-a?/c menu-item%))
```

This method returns the menu-item% object corresponding to this menu item, if it has been created (as controlled by edit-menu:create-select-all?).

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

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

Defaults to `#t`.

```
(send a-frame:standard-menus edit-menu:select-all-callback
 menu
 evt)
→ void?
 menu : (is-a?/c menu-item%)
 evt : (is-a?/c control-event%)
```

Defaults to

```
(begin (let ((edit (get-edit-target-
object))) (when (and edit (is-a? edit editor<%>)) (send edit do-
edit-operation 'select-all)))  #t)
```

```
(send a-frame:standard-menus edit-menu:select-all-on-demand item)
→ void?
 item : (is-a?/c menu-item%)
```

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

Defaults to

```
(let* ((editor (get-edit-target-object)) (enable? (and editor (is-
a? editor editor<%>) (send editor can-do-edit-
operation? 'select-all))))  (send item enable enable?))
```

```
(send a-frame:standard-menus edit-menu:select-all-string)
→ string?
```

The result of this method is used as the name of the `menu-item%`.

Defaults to (`string-constant select-all-menu-item`).

```
(send a-frame:standard-menus edit-menu:select-all-help-string)
→ string?
```

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

Defaults to (`string-constant select-all-info`).

```
(send a-frame:standard-menus edit-menu:between-select-all-and-find menu)
→ void?
 menu : (is-a?/c menu-item%)
```

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

```
(send a-frame:standard-menus edit-menu:get-find-item)
→ (or/c false/c (is-a?/c menu-item%))
```

> This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `edit-menu:create-find?`).

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

> The result of this method determines if the corresponding menu item is created. Override it to control the creation of the menu item.
>
> Defaults to `#f`.

```
(send a-frame:standard-menus edit-menu:find-callback item
                                                      control)
→ void?
  item : (is-a?/c menu-item%)
  control : (is-a?/c control-event%)
```

> Defaults to
>
> ```
> (void)
> ```

```
(send a-frame:standard-menus edit-menu:find-on-demand item)
→ void?
  item : (is-a?/c menu-item%)
```

> The menu item's on-demand proc calls this method.
>
> Defaults to
>
> ```
> (send item enable (let ((target (get-edit-target-
> object))) (and target (is-a? target editor<%>))))
> ```

```
(send a-frame:standard-menus edit-menu:find-string) → string?
```

> The result of this method is used as the name of the `menu-item%`.
>
> Defaults to `(string-constant find-menu-item)`.

```
(send a-frame:standard-menus edit-menu:find-help-string)
→ string?
```

> The result of this method is used as the help string when the `menu-item%` object is created.
>
> Defaults to `(string-constant find-info)`.

```
(send a-frame:standard-menus edit-menu:get-find-next-item)
→ (or/c false/c (is-a?/c menu-item%))
```

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `edit-menu:create-find-next?`).

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

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

Defaults to `#f`.

`(send a-frame:standard-menus edit-menu:find-next-callback`
`  item`
`  control)`
`→ void?`
`  item : (is-a?/c menu-item%)`
`  control : (is-a?/c control-event%)`

Defaults to

`    (void)`

`(send a-frame:standard-menus edit-menu:find-next-on-demand item)`
`→ void?`
`  item : (is-a?/c menu-item%)`

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

Defaults to

`    (send item enable (let ((target (get-edit-target-`
`    object))) (and target (is-a? target editor<%>))))`

`(send a-frame:standard-menus edit-menu:find-next-string)`
`→ string?`

The result of this method is used as the name of the `menu-item%`.

Defaults to `(string-constant find-next-menu-item)`.

`(send a-frame:standard-menus edit-menu:find-next-help-string)`
`→ string?`

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

Defaults to `(string-constant find-next-info)`.

`(send a-frame:standard-menus edit-menu:get-find-previous-item)`
`→ (or/c false/c (is-a?/c menu-item%))`

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `edit-menu:create-find-previous?`).

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

> The result of this method determines if the corresponding menu item is created. Override it to control the creation of the menu item.
>
> Defaults to `#f`.

```
(send a-frame:standard-menus edit-menu:find-previous-callback
  item
  control)
→ void?
  item : (is-a?/c menu-item%)
  control : (is-a?/c control-event%)
```

> Defaults to
>
> > `(void)`

```
(send a-frame:standard-menus edit-menu:find-previous-on-demand item)
→ void?
  item : (is-a?/c menu-item%)
```

> The menu item's on-demand proc calls this method.
>
> Defaults to
>
> > `(send item enable (let ((target (get-edit-target-object))) (and target (is-a? target editor<%>))))`

```
(send a-frame:standard-menus edit-menu:find-previous-string)
→ string?
```

> The result of this method is used as the name of the `menu-item%`.
>
> Defaults to `(string-constant find-previous-menu-item)`.

```
(send a-frame:standard-menus edit-menu:find-previous-help-string)
→ string?
```

> The result of this method is used as the help string when the `menu-item%` object is created.
>
> Defaults to `(string-constant find-previous-info)`.

```
(send a-frame:standard-menus edit-menu:get-show/hide-replace-item)
→ (or/c false/c (is-a?/c menu-item%))
```

> This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `edit-menu:create-show/hide-replace?`).

```
(send a-frame:standard-menus edit-menu:create-show/hide-replace?)
→ boolean?
```

> The result of this method determines if the corresponding menu item is created.
> Override it to control the creation of the menu item.
>
> Defaults to `#f`.

```
(send a-frame:standard-menus edit-menu:show/hide-replace-callback
  item
  control)
→ void?
  item : (is-a?/c menu-item%)
  control : (is-a?/c control-event%)
```

> Defaults to
>
> `(void)`

```
(send a-frame:standard-menus edit-menu:show/hide-replace-on-demand menu-item)
→ void?
  menu-item : (is-a?/c menu-item%)
```

> The menu item's on-demand proc calls this method.
>
> Defaults to
>
> `(void)`

```
(send a-frame:standard-menus edit-menu:show/hide-replace-string)
→ string?
```

> The result of this method is used as the name of the `menu-item%`.
>
> Defaults to `(string-constant show-replace-menu-item)`.

```
(send a-frame:standard-menus edit-menu:show/hide-replace-help-string)
→ string?
```

> The result of this method is used as the help string when the `menu-item%` object
> is created.
>
> Defaults to `(string-constant show/hide-replace-info)`.

```
(send a-frame:standard-menus edit-menu:get-replace-item)
→ (or/c false/c (is-a?/c menu-item%))
```

> This method returns the `menu-item%` object corresponding to this menu item,
> if it has been created (as controlled by `edit-menu:create-replace?`).

```
(send a-frame:standard-menus edit-menu:create-replace?)
→ boolean?
```

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

Defaults to `#f`.

```
(send a-frame:standard-menus edit-menu:replace-callback
  item
  control)
→ void?
  item : (is-a?/c menu-item%)
  control : (is-a?/c control-event%)
```

Defaults to

```
(void)
```

```
(send a-frame:standard-menus edit-menu:replace-on-demand menu-item)
→ void?
  menu-item : (is-a?/c menu-item%)
```

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

Defaults to

```
(void)
```

```
(send a-frame:standard-menus edit-menu:replace-string)
→ string?
```

The result of this method is used as the name of the `menu-item%`.

Defaults to `(string-constant replace-menu-item)`.

```
(send a-frame:standard-menus edit-menu:replace-help-string)
→ string?
```

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

Defaults to `(string-constant replace-info)`.

```
(send a-frame:standard-menus edit-menu:get-replace-all-item)
→ (or/c false/c (is-a?/c menu-item%))
```

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `edit-menu:create-replace-all?`).

```
(send a-frame:standard-menus edit-menu:create-replace-all?)
→ boolean?
```

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

Defaults to `#f`.

```
 (send a-frame:standard-menus edit-menu:replace-all-callback
  item
  control)
→ void?
  item : (is-a?/c menu-item%)
  control : (is-a?/c control-event%)
```

Defaults to

```
    (void)
```

```
(send a-frame:standard-menus edit-menu:replace-all-on-demand menu-item)
 → void?
  menu-item : (is-a?/c menu-item%)
```

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

Defaults to

```
    (void)
```

```
(send a-frame:standard-menus edit-menu:replace-all-string)
 → string?
```

The result of this method is used as the name of the `menu-item%`.

Defaults to (`string-constant replace-all-menu-item`).

```
(send a-frame:standard-menus edit-menu:replace-all-help-string)
 → string?
```

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

Defaults to (`string-constant replace-all-info`).

```
(send a-frame:standard-menus edit-menu:get-find-case-sensitive-item)
 → (or/c false/c (is-a?/c menu-item%))
```

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `edit-menu:create-find-case-sensitive?`).

```
(send a-frame:standard-menus edit-menu:create-find-case-sensitive?)
 → boolean?
```

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

Defaults to `#f`.

```
(send a-frame:standard-menus edit-menu:find-case-sensitive-callback
 item
 control)
→ void?
 item : (is-a?/c menu-item%)
 control : (is-a?/c control-event%)
```

Defaults to

```
(void)
```

```
(send a-frame:standard-menus edit-menu:find-case-sensitive-on-demand item)
→ void?
 item : (is-a?/c menu-item%)
```

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

Defaults to

```
(send item enable (let ((target (get-edit-target-
object))) (and target (is-a? target editor<%>))))
```

```
(send a-frame:standard-menus edit-menu:find-case-sensitive-string)
→ string?
```

The result of this method is used as the name of the menu-item%.

Defaults to (string-constant find-case-sensitive-menu-item).

```
(send a-frame:standard-menus edit-menu:find-case-sensitive-help-string)
→ string?
```

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

Defaults to (string-constant find-case-sensitive-info).

```
(send a-frame:standard-menus edit-menu:between-find-and-preferences menu)
→ void?
 menu : (is-a?/c menu-item%)
```

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

```
(send a-frame:standard-menus edit-menu:get-preferences-item)
→ (or/c false/c (is-a?/c menu-item%))
```

This method returns the menu-item% object corresponding to this menu item,
if it has been created (as controlled by edit-menu:create-preferences?).

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

$\rightarrow$ `boolean?`

> The result of this method determines if the corresponding menu item is created. Override it to control the creation of the menu item.
>
> Defaults to `(not (current-eventspace-has-standard-menus?))`.

```
(send a-frame:standard-menus edit-menu:preferences-callback
 item
 control)
→ void?
 item : (is-a?/c menu-item%)
 control : (is-a?/c control-event%)
```

> Defaults to
>
> `(begin (preferences:show-dialog) #t)`

```
(send a-frame:standard-menus edit-menu:preferences-on-demand menu-item)
→ void?
 menu-item : (is-a?/c menu-item%)
```

> The menu item's on-demand proc calls this method.
>
> Defaults to
>
> `(void)`

```
(send a-frame:standard-menus edit-menu:preferences-string)
→ string?
```

> The result of this method is used as the name of the `menu-item%`.
>
> Defaults to `(string-constant preferences-menu-item)`.

```
(send a-frame:standard-menus edit-menu:preferences-help-string)
→ string?
```

> The result of this method is used as the help string when the `menu-item%` object is created.
>
> Defaults to `(string-constant preferences-info)`.

```
(send a-frame:standard-menus edit-menu:after-preferences menu)
→ void?
 menu : (is-a?/c menu-item%)
```

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

```
(send a-frame:standard-menus help-menu:before-about menu)
→ void?
 menu : (is-a?/c menu-item%)
```

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

```
(send a-frame:standard-menus help-menu:get-about-item)
 → (or/c false/c (is-a?/c menu-item%))
```

This method returns the `menu-item%` object corresponding to this menu item, if it has been created (as controlled by `help-menu:create-about?`).

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

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

Defaults to `#f`.

```
(send a-frame:standard-menus help-menu:about-callback item
                                                      control)
 → void?
  item : (is-a?/c menu-item%)
  control : (is-a?/c control-event%)
```

Defaults to

```
(void)
```

```
(send a-frame:standard-menus help-menu:about-on-demand menu-item)
 → void?
  menu-item : (is-a?/c menu-item%)
```

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

Defaults to

```
(void)
```

```
(send a-frame:standard-menus help-menu:about-string) → string?
```

The result of this method is used as the name of the `menu-item%`.

Defaults to `(string-constant about-menu-item)`.

```
(send a-frame:standard-menus help-menu:about-help-string)
 → string?
```

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

Defaults to `(string-constant about-info)`.

```
(send a-frame:standard-menus help-menu:after-about menu)
 → void?
```

menu : `(is-a?/c menu-item%)`

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

`frame:standard-menus-mixin` : `(class? . -> . class?)`
  argument extends/implements: `frame:basic<%>`
  result implements: `frame:standard-menus<%>`

The result of this mixin implements `frame:standard-menus<%>`.

`(send` *a-frame:standard-menus* `on-close)` → `void`

> Augments `on-close` in `top-level-window<%>`.

> Removes the preferences callbacks for the menu items

`frame:editor<%>` : `interface?`
  implements: `frame:standard-menus<%>`

Frame classes matching this interface support embedded editors.

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

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

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

> This returns the prefix for the frame's label.

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

> Sets the prefix for the label of the frame.

`(send` *a-frame:editor* `get-canvas%)`
→ `(subclass?/c editor-canvas%)`

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

> Returns `editor-canvas%`.

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

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

(send *a-frame:editor* get-editor%) → (is-a?/c editor<%>)

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

Override this method to specify a different editor class.

Returns the value of the init-field editor%.

(send *a-frame:editor* get-editor<%>) → interface

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

Returns editor<%>.

(send *a-frame:editor* make-editor)
→ (instance (is-a?/c 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.

Calls (make-object get-editor%).

(send *a-frame:editor* revert) → void

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.

(send *a-frame:editor* save [*format*]) → boolean?
  format : (or/c 'guess 'standard 'text 'text-force-cr 'same 'copy)
          = 'same

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

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.

(send *a-frame:editor* save-as [*format*]) → boolean?
  format : (or/c 'guess 'standard 'text 'text-force-cr 'same 'copy)
          = 'same

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

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

(send *a-frame:editor* get-canvas)
→ (instance (subclass?/c canvas%))

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

```
(send a-frame:editor get-editor)
 → (instance (is-a?/c editor<%>))
```

Returns the editor in this frame.

```
frame:editor-mixin : (class? . -> . class?)
```
  argument extends/implements: `frame:standard-menus<%>`
  result 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, a 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.

```
 (new frame:editor-mixin
   [filename filename]
   [editor% editor%]
  [[parent parent]
   [width width]
   [height height]
   [x x]
   [y y]
   [style style]
   [enabled enabled]
   [border border]
   [spacing spacing]
   [alignment alignment]
   [min-width min-width]
   [min-height min-height]
   [stretchable-width stretchable-width]
   [stretchable-height stretchable-height]])
 → (is-a?/c frame:editor-mixin)
 filename : string?
 editor% : (is-a?/c editor<%>)
 parent : (or/c (is-a?/c frame%) false/c) = #f
 width : (or/c (integer-in 0 10000) false/c) = #f
 height : (or/c (integer-in 0 10000) false/c) = #f
 x : (or/c (integer-in -10000 10000) false/c) = #f
 y : (or/c (integer-in -10000 10000) false/c) = #f
 style : (listof (or/c 'no-resize-border 'no-caption 'no-system-menu 'hide-menu-bar 'mdi
         = null
```

```
enabled : any/c = #t
border : (integer-in 0 1000) = 0
spacing : (integer-in 0 1000) = 0
alignment : (list/c (or/c 'left  'center 'right)  (or/c 'top 'center 'bottom))
           = '(center top)
min-width : (integer-in 0 10000) = graphical-minimum-width
min-height : (integer-in 0 10000) = graphical-minimum-height
stretchable-width : any/c = #t
stretchable-height : any/c = #t
```

`(send a-frame:editor get-filename)` → `(or/c #f path)`

Overrides `get-filename` in `frame:basic<%>`.

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

`(send a-frame:editor editing-this-file? filename)` → `boolean?`
  `filename : path`

Overrides `editing-this-file?` in `frame:basic<%>`.

Returns `#t` if the filename is the file that this frame is editing.

`(send a-frame:editor on-close)` → `void`

Augments `on-close` in `frame:standard-menus<%>`.

Calls the `editor:basic<%>`'s method `on-close`.

`(send a-frame:editor can-close?)` → `void`

Augments `can-close?` in `top-level-window<%>`.

Calls the `editor:basic<%>`'s method `can-close?`.

`(send a-frame:editor get-label)` → `string`

Overrides `get-label` in `window<%>`.

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

`(send a-frame:editor set-label label)` → `void`
  `label : string?`

Overrides `set-label` in `window<%>`.

Sets the label, but preserves the label's prefix. See also `set-label-prefix`.

`(send a-frame:editor file-menu:open-callback item`
                                              `evt)` → `void`
  `item : any`
  `evt : mouse-event`

82

Overrides `file-menu:open-callback` in `frame:standard-menus<%>`.

Calls `handler:open-file` with the directory of the saved file associated with this editor (if any).

(send `a-frame:editor` `file-menu:revert-on-demand`) → `void`

Overrides `file-menu:revert-on-demand` in `frame:standard-menus<%>`.

Disables the menu item when the editor is locked.

(send `a-frame:editor` `file-menu:revert-callback` *item*
                                                    evt) → `void`
  item : `(is-a?/c menu-item%)`
  evt : `(is-a?/c control-event%)`

Overrides `file-menu:revert-callback` in `frame:standard-menus<%>`.

Informs the user that this action is not undoable and, if they still want to continue, calls `revert`.

(send `a-frame:editor` `file-menu:create-revert?`) → `boolean?`

Overrides `file-menu:create-revert?` in `frame:standard-menus<%>`.

returns #t

(send `a-frame:editor` `file-menu:save-callback` *item*
                                                 evt) → `void`
  item : `(is-a?/c menu-item%)`
  evt : `(is-a?/c control-event%)`

Overrides `file-menu:save-callback` in `frame:standard-menus<%>`.

Saves the file in the editor.

(send `a-frame:editor` `file-menu:create-save?`) → `boolean?`

Overrides `file-menu:create-save?` in `frame:standard-menus<%>`.

returns #t

(send `a-frame:editor` `file-menu:save-as-callback` *item*
                                                    evt) → `void`
  item : `(is-a?/c menu-item%)`
  evt : `(is-a?/c control-event%)`

Overrides `file-menu:save-as-callback` in `frame:standard-menus<%>`.

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

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

Overrides `file-menu:create-save-as?` in `frame:standard-menus<%>`.

returns #t

```
(send a-frame:editor file-menu:print-callback item
                                                evt) → void
  item : (is-a?/c menu-item%)
  evt : (is-a?/c control-event%)
```

Overrides `file-menu:print-callback` in `frame:standard-menus<%>`.

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`.

```
(send a-frame:editor file-menu:create-print?) → boolean?
```

Overrides `file-menu:create-print?` in `frame:standard-menus<%>`.

returns #t

```
(send a-frame:editor file-menu:between-save-as-and-print file-menu)
 → void
  file-menu : (is-a?/c menu%)
```

Overrides `file-menu:between-save-as-and-print` in `frame:standard-menus<%>`.

Creates a Print Setup menu item.

```
(send a-frame:editor edit-menu:between-select-all-and-find edit-menu)
 → void
  edit-menu : (instance menu%)
```

Overrides `edit-menu:between-select-all-and-find` in `frame:standard-menus<%>`.

Adds a menu item for toggling `auto-wrap` in the focused text.

```
(send a-frame:editor help-menu:about-callback item
                                                evt) → void
  item : (is-a?/c menu-item%)
  evt : (is-a?/c control-event%)
```

Overrides `help-menu:about-callback` in `frame:standard-menus<%>`.

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

```
(send a-frame:editor help-menu:about-string) → string
```

Overrides `help-menu:about-string` in `frame:standard-menus<%>`.

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

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

    Overrides `help-menu:create-about?` in `frame:standard-menus<%>`.

    returns #t

`frame:open-here<%>` : `interface?`
  implements: `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.

(send *a-frame:open-here* `get-open-here-editor`)
→ (`is-a?/c editor<%>`)

    When the user switches the visible file in this frame, the of this method is the editor that gets switched.

    Defaultly returns the result of `get-editor`.

(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).

`frame:open-here-mixin` : (`class? . -> . class?`)
  argument extends/implements: `frame:editor<%>`
  result implements: `frame:open-here<%>`

Provides an implementation of `frame:open-here<%>`

(send *a-frame:open-here* `file-menu:new-on-demand` *item*) → `void`
  item : (`is-a?/c menu-item%`)

    Overrides `file-menu:new-on-demand` in `frame:standard-menus<%>`.

    Sets the label of *item* to `"New..."` if the preference `'framework:open-here?` is set.

(send *a-frame:open-here* `file-menu:new-callback` *item*
                                     evt) → `void`

```
item : (instance (subclass?/c menu-item%))
evt : (instance control-event%)
```

> Overrides `file-menu:new-callback` in `frame:standard-menus<%>`.

> 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.

```
(send a-frame:open-here file-menu:open-on-demand item) → void
  item : (is-a?/c menu-item%)
```

> Overrides `file-menu:open-on-demand` in `frame:standard-menus<%>`.

> Sets the label of *item* to "Open Here..." if the preference `'framework:open-here?` is set.

```
(send a-frame:open-here on-close) → void
```

> Augments `on-close` in `frame:standard-menus<%>`.

> Calls `set-open-here-frame` with `#f` if the result of `get-open-here-frame` is `eq?` to `this`.

```
(send a-frame:open-here on-activate on?) → void
  on? : boolean?
```

> Overrides `on-activate` in `top-level-window<%>`.

> When *on?* is `#t`, calls `set-open-here-frame` with `this`.

```
frame:text<%> : interface?
  implements: frame:editor<%>
```

Frames matching this interface provide support for `text%`s.

```
frame:text-mixin : (class? . -> . class?)
  argument extends/implements: frame:editor<%>
  result implements: frame:text<%>
```

This mixins adds support for `text%`s in the frame.

```
(new frame:text-mixin [editor% editor%])
 → (is-a?/c frame:text-mixin)
```

editor% : (extends `text%`)

> Calls the super initialization with either the value of the *editor%* init or, if none was supplied, it passes `text%`.

(send *a-frame:text* `get-editor<%>`) → `interface`

> Overrides `get-editor<%>` in `frame:editor<%>`.
>
> Returns (`class->interface text%`).

`frame:pasteboard<%>` : `interface?`
  implements: `frame:editor<%>`

Frames matching this interface provide support for `pasteboard%`s.

`frame:pasteboard-mixin` : (`class? . -> . class?`)
  argument extends/implements: `frame:editor<%>`
  result implements: `frame:pasteboard<%>`

This mixin provides support for pasteboards in a frame.

(new `frame:pasteboard-mixin` [editor% *editor%*])
 → (`is-a?/c frame:pasteboard-mixin`)
  editor% : (extends `pasteboard%`)

> Calls the super initialization with either the value of the *editor%* init or, if none was supplied, it passes `pasteboard%`.

(send *a-frame:pasteboard* `get-editor<%>`) → `interface`

> Overrides `get-editor<%>` in `frame:editor<%>`.
>
> Returns (`class->interface pasteboard%`).

`frame:delegate<%>` : `interface?`
  implements: `frame:status-line<%>`
              `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.

```
(send a-frame:delegate get-delegated-text)
→ (instanceof (is-a?/c text:delegate<%>))
```

Returns the delegate text.

```
(send a-frame:delegate delegated-text-shown?) → boolean?
```

Returns `#t` if the delegate is visible, and `#f` if it isn't.

```
(send a-frame:delegate hide-delegated-text) → void
```

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 show-delegated-text) → void
```

Makes the delegated text visible.

When the delegated text is shown, the `set-delegate` method of `get-delegated-text`is called with the text to delegate messages to.

See also `hide-delegated-text`.

```
(send a-frame:delegate delegate-moved) → void
```

This method is called when the visible region of the delegate editor changes, so that the blue region in the delegatee is updated.

```
frame:delegate-mixin : (class? . -> . class?)
   argument extends/implements: frame:status-line<%>
                                frame:text<%>
   result implements: frame:delegate<%>
```

Adds support for a 20,000-feet view via `text:delegate<%>` and `text:delegate-mixin`

```
(send a-frame:delegate make-root-area-container class
                                                parent)
→ (is-a?/c panel%)
  class : (subclass?/c panel%)
  parent : (instanceof (subclass?/c panel%))
```

Overrides `make-root-area-container` in `frame:basic<%>`.

adds a panel outside to hold the delegate `editor-canvas%` and `text%`.

(send *a-frame:delegate* get-editor<%>) → interface

> Overrides get-editor<%> in frame:editor<%>.
>
> Returns text:delegate.

(send *a-frame:delegate* get-editor%)
 → (is-a?/c text:delegate<%>)

> Overrides get-editor% in frame:editor<%>.
>
> returns the super result, with the text:delegate-mixin mixed in.

---

frame:searchable<%> : interface?
  implements: frame:basic<%>

Frames that implement this interface support searching.

(send *a-frame:searchable* search *direction*) → void?
  direction : (symbols 'forward 'backward)

> Searches for the text in the search edit in the result of get-text-to-search.
>
> If the text is found and it sets the selection to the found text.

(send *a-frame:searchable* search-replace) → boolean?

> If there is a dark purple bubble (ie, if the replace portion of the search bar is visible and there is a search hit after the insertion point), then this will replace it with the contents of the replace editor and move the insertion point to just after that, or to the end of the editor (if there are no more search hits after the insertion point, but there are search hits before it).

(send *a-frame:searchable* replace-all) → void?

> Loops through the text from the beginning to the end, replacing all occurrences of the search string with the contents of the replace edit.

(send *a-frame:searchable* get-text-to-search) → (is-a?/c text%)

> Returns the last value passed to set-text-to-search.

(send *a-frame:searchable* set-text-to-search *txt*) → void?
  txt : (or/c false/c (is-a?/c (subclass?/c text%)))

> Sets the current text to be searched.

(send *a-frame:searchable* search-hidden?) → boolean?

Returns `#t` if the search subwindow is visiable and `#f` otherwise.

`(send` `a-frame:searchable` `hide-search)` → `void?`

This method hides the searching information on the bottom of the frame.

`(send` `a-frame:searchable` `unhide-search` `move-focus?)` → `void?`
  `move-focus?` : `boolean?`

When the searching sub window is hidden, makes it visible. If `move-focus?` is `#f`, the focus is not moved, but if it is any other value, the focus is moved to the find window.

`(send` `a-frame:searchable` `get-case-sensitive-search?)`
 → `boolean?`

Returns `#t` if the search is currently case-sensitive. (This method's value depends on the preference `'framework:case-sensitive-search?`, but the preference is only consulted when the frame is created.)

`(send` `a-frame:searchable` `search-hits-changed)` → `void?`

This method is called when the number of search matches changes and it updates the GUI.

`frame:searchable-mixin` : `(class? . -> . class?)`
  argument extends/implements: `frame:standard-menus<%>`
  result implements: `frame:searchable<%>`

This mixin adds support for searching in the `editor<%>` in this frame.

`(send` `a-frame:searchable` `edit-menu:find-callback)` → `boolean?`

Overrides `edit-menu:find-callback` in `frame:standard-menus<%>`.

Toggles the focus between the find window and the window being searched. When moving to the window with the search string, selects the entire range in the buffer.

`(send` `a-frame:searchable` `edit-menu:create-find?)` → `boolean?`

Overrides `edit-menu:create-find?` in `frame:standard-menus<%>`.

returns `#t`.

`(send` `a-frame:searchable` `edit-menu:find-again-callback` `item`
                                                            `evt)`
 → `void?`

```
item : (is-a?/c menu-item%)
evt : (is-a?/c control-event%)
```

Overrides <method not found>.

Calls unhide-search and then `search`.

```
(send a-frame:searchable edit-menu:create-find-again?)
 → boolean?
```

Overrides <method not found>.

returns `#t`.

```
(send a-frame:searchable edit-menu:find-again-backwards-callback
 item
 evt)
 → void?
 item : (is-a?/c menu-item%)
 evt : (is-a?/c control-event%)
```

Overrides <method not found>.

Calls unhide-search and then `search`.

```
(send a-frame:searchable edit-menu:create-find-again-backwards?)
 → boolean?
```

Overrides <method not found>.

returns `#t`.

```
(send a-frame:searchable edit-menu:replace-all-callback)
 → boolean?
```

Overrides `edit-menu:replace-all-callback` in `frame:standard-menus<%>`.

Calls `replace-all`.

```
(send a-frame:searchable edit-menu:replace-all-on-demand item)
 → void
 item : menu-item%
```

Overrides `edit-menu:replace-all-on-demand` in `frame:standard-menus<%>`.

Disables *item* when `search-hidden?` returns `#t` and enables it when that method returns `#f`.

```
(send a-frame:searchable edit-menu:create-replace-all?)
 → boolean?
```

Overrides `edit-menu:create-replace-all?` in `frame:standard-menus<%>`.

returns `#t`.

(send `a-frame:searchable` `edit-menu:find-case-sensitive-callback`)
→ `boolean?`

> Overrides `edit-menu:find-case-sensitive-callback` in `frame:standard-menus<%>`.

> Updates the state of the case-sensitive searching for this frame, and sets the `'framework:case-sensitive-search?` preference for later frames.

(send `a-frame:searchable` `edit-menu:find-case-sensitive-on-demand` `item`)
→ `void`
  `item` : `menu-item%`

> Overrides `edit-menu:find-case-sensitive-on-demand` in `frame:standard-menus<%>`.

> Checks `item` when searching is case-sensitive and unchecks it otherwise.

(send `a-frame:searchable` `edit-menu:create-find-case-sensitive?`)
→ `boolean?`

> Overrides `edit-menu:create-find-case-sensitive?` in `frame:standard-menus<%>`.

> returns `#t`.

(send `a-frame:searchable` `make-root-area-container`)
→ `(is-a?/c area-container<%>)`

> Overrides `make-root-area-container` in `frame:basic<%>`.

> Builds a panel for the searching information.

(send `a-frame:searchable` `on-close`) → `void`

> Augments `on-close` in `frame:standard-menus<%>`.

> Cleans up after the searching frame.

`frame:searchable-text<%>` : `interface?`
  implements: `frame:searchable<%>`
              `frame:text<%>`

`frame:searchable-text-mixin` : `(class? . -> . class?)`
  argument extends/implements: `frame:text<%>`
                               `frame:searchable<%>`

92

result implements: `frame:searchable-text<%>`

`(send` *a-frame:searchable-text* `get-text-to-search)`
→ `(instanceof text%)`

Overrides `get-text-to-search` in `frame:searchable<%>`. This method is final, so it cannot be overiddden.

Returns the result of `get-editor`.

`(send` *a-frame:searchable-text* `get-editor<%>)`
→ `(is-a?/c editor<%>)`

Overrides `get-editor<%>` in `frame:editor<%>`.

Returns `text:searching<%>`.

`(send` *a-frame:searchable-text* `get-editor%)`
→ `(is-a?/c editor<%>)`

Overrides `get-editor%` in `frame:editor<%>`.

Returns `(text:searching-mixin (super get-editor%))`.

`frame:basic%` : `class?`
  superclass: `(frame:register-group-mixin (frame:basic-mixin frame%))`

`frame:size-pref%` : `class?`
  superclass: `(frame:size-pref-mixin frame:basic%)`

`frame:info%` : `class?`
  superclass: `(frame:info-mixin frame:basic%)`

`frame:text-info%` : `class?`
  superclass: `(frame:text-info-mixin frame:info%)`

93

```
frame:pasteboard-info% : class?
  superclass: (frame:pasteboard-info-mixin frame:text-info%)
```

```
frame:status-line% : class?
  superclass: (frame:status-line-mixin frame:text-info%)
```

```
frame:standard-menus% : class?
  superclass: (frame:standard-menus-mixin frame:status-line%)
```

```
frame:editor% : class?
  superclass: (frame:editor-mixin frame:standard-menus%)
```

```
frame:open-here% : class?
  superclass: (frame:open-here-mixin frame:editor%)
```

```
frame:text% : class?
  superclass: (frame:text-mixin frame:open-here%)
```

```
frame:searchable% : class?
  superclass: (frame:searchable-text-mixin (frame:searchable-mixin frame:text%))
```

`frame:delegate%` : `class?`
  superclass: `(frame:delegate-mixin frame:searchable%)`

`frame:pasteboard%` : `class?`
  superclass: `(frame:pasteboard-mixin frame:open-here%)`

```
(frame:setup-size-pref  size-pref-sym
                        width
                        height)          → void
  size-pref-sym : symbol?
  width : number?
  height : number?
```

Initializes a preference for the `frame:size-pref` mixin.

The first argument should be the preferences symbol, and the second an third should be the default width and height, respectively.

```
(frame:add-snip-menu-items  menu
                            menu-item%
                            [func])      → void?
  menu : (is-a?/c menu%)
  menu-item% : (subclass?/c menu-item%)
  func : (-> (is-a?/c menu-item%) void?) = void
```

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.

Calls *func* right after inserting each menu item.

```
(frame:reorder-menus frame) → void?
  frame : (is-a?/c frame%)
```

Re-orders the menus in a frame. It moves the "File" and "Edit" menus to the front of the menubar and moves the "Windows" and "Help" menus to the end of the menubar.

This is useful in conjunction with the frame classes. After instantiating the class and adding ones own menus, the menus will be mis-ordered. This function fixes them up.

```
(frame:remove-empty-menus frame) → void?
  frame : (is-a?/c frame%)
```

Removes empty menus in a frame.

```
(frame:current-icon) → (or/c #f
                              (is-a?/c bitmap%)
                              (cons/c (is-a?/c bitmap%)
                                      (is-a?/c bitmap%)))
(frame:current-icon icon-spec) → void?
  icon-spec : (or/c #f
                    (is-a?/c bitmap%)
                    (cons/c (is-a?/c bitmap%)
                            (is-a?/c bitmap%)))
```

The value of this parameter is used by the initialization code of `frame:basic-mixin`.

- If it is `#f`, then its value is ignored.

- It it is a `bitmap%`, then the `set-icon` is called with the bitmap, the result of invoking the `bitmap%` `get-loaded-mask` method, and `'both`.

- If it is a pair of bitmaps, then the `set-icon` method is invoked twice, once with each bitmap in the pair. The first bitmap is passed (along with the result of its `bitmap%` `get-loaded-mask`) and `'small`, and then the second bitmap is passed (also along with the result of its `bitmap%` `get-loaded-mask`) and `'large`.

Defaults to `#f`.

13 Group

`group:%` : `class?`
  superclass: `object%`

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`.

(send *a-group:* `set-open-here-frame` *frame*) → `void`
  frame : `(is-a?/c frame:editor%)`

     Sets the frame to load new files into. See also `frame:open-here<%>`.

(send *a-group:* `get-open-here-frame`)
→ `(or/c false/c (is-a?/c frame:editor<%>))`

     Returns the currently saved frame to load new files into.

(send *a-group:* `get-mdi-parent`)
→ `(or/c false/c (instance frame%))`

     The result of this method must be used as the parent frame for each frame in the group.

(send *a-group:* `get-frames`)
→ `(list-of (instance frame:basic<%>))`

     Returns the frames in the group.

(send *a-group:* `frame-label-changed` *frame*) → `void`
  frame : `(is-a?/c frame:basic<%>)`

     This method is called by frames constructed with `frame:basic-mixin` when their titles change.

     Updates the windows menu of each frame in the group.

(send *a-group:* `frame-shown/hidden`) → `void`

     This method is called by instances of `frame:basic%` to notify the frame group that a frame's visibility is changed.

     Updates the Windows menus of all of the frames in the frame group.

(send *a-group:* `for-each-frame` *f*) → `void`

`f` : `((instance frame:basic<%>) -> void)`

> 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`.
>
> Applies `f` to each frame in the group

`(send a-group: get-active-frame)` → `(is-a?/c frame:basic<%>)`

> Returns the frame with the keyboard focus or the first frame in the group.

`(send a-group: set-active-frame frame)` → `void`
  `frame` : `(is-a?/c frame:basic<%>)`

> Sets the active frame in the group. This method is called by `on-activate`.

`(send a-group: insert-frame frame)` → `void`
  `frame` : `(is-a?/c frame:basic<%>)`

> Inserts a frame into the group.

`(send a-group: remove-frame frame)` → `void`
  `frame` : `(is-a?/c frame:basic<%>)`

> Removes a frame from the group.

`(send a-group: clear)` → `boolean?`

> 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: on-close-all)` → `void`

> 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`.
>
> Calls the `on-close` method and the `show` method (with `#f` as argument) on each frame in the group.

`(send a-group: can-close-all?)` → `void`

> 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.
>
> Calls the `can-close?` method of each frame in the group.

`(send a-group: locate-file name)`

`→ (or/c false/c (is-a?/c frame:basic<%>))`
  `name : path?`

 Returns the frame that is editing or viewing the file *name*.

`(group:get-the-frame-group)` → `(is-a?/c group:%)`

This returns the frame group.

`(group:on-close-action)` → `void?`

See also `group:can-close-check`.

Call this function from the `can-close?` callback of a frame in order for the group to properly close the application.

`(group:can-close-check)` → `boolean?`

See also `group:on-close-action`.

Call this function from the `can-close?` callback of a frame in order for the group to properly close the application.

`(group:add-to-windows-menu proc)` → `any`
  `proc : (-> (is-a?/c menu%) any)`

Procedures passed to this function are called when the Windows menu is created. Use it to add additional menu items.

14   GUI Utilities

```
(require framework/gui-utils)
```

```
(gui-utils:trim-string str size)
 → (and/c string?
           (λ (str)
             ((string-length str) . <= . size)))
  str : string?
  size : (and/c number? positive?)
```

Constructs a string whose size is less than *size* by trimming the *str* and inserting an ellispses into it.

```
(gui-utils:quote-literal-label str)
 → (and/c string?
           (lambda (str)
             ((string-length str) . <= . 200)))
  str : string?
```

Constructs a string whose ampersand characters are escaped; the label is also trimmed to <= 200 characters.

```
(gui-utils:format-literal-label str
                                rest ...)
 → (and/c string?
           (lambda (str)
             ((string-length str) . <= . 200)))
  str : string?
  rest : (listof any/c)
```

Formats a string whose ampersand characters are mk-escaped; the label is also trimmed to <= 200 mk-characters.

```
(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.

Just returns what system-position-ok-before-cancel? does.

See also gui-utils:ok/cancel-buttons.

```
(gui-utils:ok/cancel-buttons  parent
                              confirm-callback
                              cancel-callback
                             [confirm-label
                              cancel-label])
 → (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? = (string-constant ok)
  cancel-label : string? = (string-constant cancel)
```

Adds an Ok and a cancel button to a panel, changing the order to suit the platform. Under Mac OS X 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: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: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:show-busy-cursor thunk [delay]) → any/c
  thunk : (-> any/c)
  delay : integer? = (gui-utils:cursor-delay)
```

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: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
        (λ () .. init watch cursor ...)
        (λ () .. close watch cursor ...))])

  (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:local-busy-cursor  window
                              thunk
                             [delay]) → any/c
  window : (is-a?/c window<%>)
  thunk : (-> any/c)
  delay : integer? = (gui-utils:cursor-delay)
```

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:unsaved-warning  filename
                            action
                            [can-save-now?
                            parent
                            cancel?])
 → (symbols 'continue 'save 'cancel)
  filename : string?
  action : string?
  can-save-now? : boolean? = #f
  parent : (or/c false/c              = #f
               (is-a?/c frame%)
               (is-a?/c dialog%))
  cancel? : boolean? = #t
```

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`.

If `cancel?` is `#t` there is a cancel button in the dialog and the result may be `'cancel`. If it is `#f`, then there is no cancel button, and `'cancel` will not be the result of the function.

```
(gui-utils:get-choice  message
                       true-choice
                       false-choice
                       [title
                       default-result
                       parent
                       style
                       checkbox-proc
                       checkbox-label]) → any/c
  message : string?
  true-choice : string?
  false-choice : string?
  title : string? = (string-constant warning)
  default-result : any/c = 'disallow-close
  parent : (or/c false/c (is-a?/c frame%) (is-a?/c dialog%))
         = #f
  style : (symbols 'app 'caution 'stop) = 'app
  checkbox-proc : (or/c false/c (case-> (boolean? . -> . void?)
                                        (-> boolean?)))
              = #f
```

```
checkbox-label : string? = (string-constant dont-ask-again)
```

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.

The *style* parameter is (eventually) passed to *message* as an icon in the dialog.

If *checkbox-proc* is given, it should be a procedure that behaves like a parameter for getting/setting a boolean value. The intention for this value is that it can be used to disable the dialog. When it is given, a checkbox will appear with a *checkbox-label* label (defaults to the dont-ask-again string constant), and that checkbox value will be sent to the *checkbox-proc* when the dialog is closed. Note that the dialog will always pop-up — it is the caller's responsibility to avoid the dialog if not needed.

```
(gui-utils:get-clicked-clickback-delta [white-on-black?])
 → (is-a?/c style-delta%)
  white-on-black? : boolean? = #f
```

This delta is designed for use with set-clickback. Use it as one of the style-delta% argument to set-clickback.

If *white-on-black?* is true, the function returns a delta suitable for use on a black background.

See also gui-utils:get-clickback-delta.

```
(gui-utils:get-clickback-delta [white-on-black?])
 → (is-a?/c style-delta%)
  white-on-black? : boolean? = #f
```

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.

If *white-on-black?* is true, the function returns a delta suitable for use on a black back-

ground.

See also `gui-utils:get-clicked-clickback-delta`.

15 Handler

```
(handler:handler? obj) → boolean?
  obj : any/c
```

This predicate determines if its input is a handler.

```
(handler:handler-name handler) → string?
  handler : handler:handler?
```

Extracts the name from a handler.

```
(handler:handler-extension handler)
 → (or/c (path? . -> . boolean?) (listof string?))
  handler : handler:handler?
```

Extracts the extension from a handler.

```
(handler:handler-handler handler)
 → (path? . -> . (is-a?/c frame:editor<%>))
  handler : handler:handler?
```

Extracs the handler's handling function.

```
(handler:insert-format-handler name
                               pred
                               handler) → void?
  name : string?
  pred : (or/c string? (listof string?) (path? . -> . boolean?))
  handler : (path? . -> . (or/c false/c (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.

The most recently added format handler takes precedence over all other format handlers.

```
(handler:find-named-format-handler name)
 → (path? . -> . (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:find-format-handler filename)
 → (path? . -> . (is-a?/c frame:editor<%>))
  filename : path?
```

This function selects a format handler. See also handler:insert-format-handler.

It finds a handler based on filename.

```
(handler:edit-file filename [make-default])
 → (or/c false/c (is-a?/c frame:editor<%>))
  filename : (or/c path? false/c)
  make-default : (-> (is-a?/c frame:editor<%>))
                = (λ () ((handler:current-create-new-window) filename))
```

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:current-create-new-window)
```

107

```
  → (-> (or/c false/c path?) (is-a?/c frame%))
(handler:current-create-new-window proc) → void?
  proc : (-> (or/c false/c path?) (is-a?/c frame%))
```

This is a parameter that controls how the framework creates new application windows.

The default setting is this:

```
(λ (filename)
  (let ([frame (make-object frame:text-info-file% filename)])
    (send frame show #t)
    frame))
```

```
(handler:open-file [dir])
 → (or/c false/c (is-a?/c frame:basic<%>))
  dir : (or/c false/c path? string?) = #f
```

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`, passing along `dir`.

```
(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: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:add-to-recent filename) → void?
  filename : path?
```

Adds a filename to the list of recently opened files.

```
(handler:set-recent-position filename
                             start
                             end)      → void?
  filename : path?
  start : number?
  end : number?
```

Sets the selection of the recently opened file to `start` and `end`.

```
(handler:size-recently-opened-files num) → void?
  num : number?
```

Sizes the `'framework:recently-opened-files/pos` preference list length to *num*.

16 Icon

`(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-eof-bitmap)` → `(is-a?/c bitmap%)`

This returns the `bitmap%` used for the clickable "eof" icon from `text:ports`.

`(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-lock-bitmap)` → `(is-a?/c bitmap%)`

This returns the lock's `bitmap`.

The bitmap may not respond `#t` to the `ok?` method.

`(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-anchor-bitmap)` → `(is-a?/c bitmap%)`

This returns the anchor's `bitmap`.

The bitmap may not respond `#t` to the `ok?` method.

`(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-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.

`(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-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.

17 Keymap

`keymap:aug-keymap<%>` : `interface?`
   implements: `keymap%`

This keymap overrides some of the built in `keymap%` methods to be able to extract the key-bindings from the keymap.

`(send a-keymap:aug-keymap get-chained-keymaps)`
 `→ (listof (instance keymap%))`

>   Returns the list of keymaps that are chained to this one.

`(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.

`(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`.

`keymap:aug-keymap-mixin` : `(class? . -> . class?)`
   argument extends/implements: `keymap%`
   result implements: `keymap:aug-keymap<%>`

`(send a-keymap:aug-keymap chain-to-keymap next`
                                         `prefix?)` `→ void`
  `next` : `(instance keymap%)`
  `prefix?` : `boolean?`

>   Overrides `chain-to-keymap` in `keymap%`.

>   Keeps a list of the keymaps chained to this one.

`(send a-keymap:aug-keymap remove-chained-keymap keymap)` `→ void`
  `keymap` : `(is-a?/c keymap)`

>   Overrides `remove-chained-keymap` in `keymap%`.

Keeps the list of the keymaps chained to this one up to date.

```
(send a-keymap:aug-keymap map-function key-name
                                       function-name) → void
  key-name : string
  function-name : string
```

Overrides `map-function` in `keymap%`.

Keeps a separate record of the key names and functions that they are bound to in this keymap.

```
keymap:aug-keymap% : class?
  superclass: (keymap:aug-keymap-mixin keymap%)
```

```
(keymap:remove-user-keybindings-file user-keybindings-path)
 → any
  user-keybindings-path : any/c
```

Removes the keymap previously added by `keymap:add-user-keybindings-file`.

```
(keymap:add-user-keybindings-file user-keybindings-path-or-require-spec)
 → any
  user-keybindings-path-or-require-spec : any/c
```

Chains the keymap defined by *user-keybindings-path-or-require-spec* to the global keymap, returned by `keymap:get-global`.

If *user-keybindings-path-or-require-spec* is a path, the module is loaded directly from that path. Otherwise, *user-keybindings-path-or-require-spec* is treated like an argument to `require`.

```
(keymap:add-to-right-button-menu) → (-> (is-a?/c popup-menu%)
                                        (is-a?/c editor<%>)
                                        (is-a?/c event%)
                                        void?)
(keymap:add-to-right-button-menu proc) → void?
  proc : (-> (is-a?/c popup-menu%)
             (is-a?/c editor<%>)
             (is-a?/c event%)
             void?)
```

113

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)
 → (-> (is-a?/c popup-menu%) (is-a?/c editor<%>) (is-a?/c event%) void?)
(keymap:add-to-right-button-menu/before proc) → void?
  proc : (-> (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 thunk-proc) → any/c
  thunk-proc : (-> any/c)
```

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 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) → (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 `"a:"`; under windows the shortcut key is `"c:"` and under MacOS, the shortcut key is `"d:"`.

`(keymap:get-file)` → `(is-a?/c keymap%)`

This returns a keymap for handling file operations.

`(keymap:get-user)` → `(is-a?/c keymap%)`

This returns a keymap that contains all of the keybindings in the keymaps loaded via `keymap:add-user-keybindings-file`

`(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)` → `(is-a?/c keymap%)`

This returns a keymap for searching operations.

`(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 `"m:a"` and `"ESC;a"` to `func`.

```
(keymap:send-map-function-meta keymap
                               key
                               func) → void?
  keymap : (is-a?/c keymap%)
```

115

```
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*) → `void?`
  keymap : (`is-a?/c keymap%`)

This sets up the input keymap with the bindings described in `keymap:get-editor`.

(`keymap:setup-file` *keymap*) → `void?`
  keymap : (`is-a?/c keymap%`)

This extends a `keymap%` with the bindings for files.

(`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 as "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.

116

- 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-Backslash : "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) → void?
  keymap : (is-a?/c keymap%)
```

This extends a `keymap%` with the bindings for searching.

```
(keymap:set-chained-keymaps keymap
                            children-keymaps) → void?
  keymap : (is-a?/c keymap:aug-keymap<%>)
  children-keymaps : (listof (is-a?/c keymap%))
```

Sets *keymap*'s chained keymaps to *children-keymaps*, unchaining any keymaps that are currently chained to *keymap*.

```
(keymap:remove-chained-keymap editor
                              keymap) → void?
  editor : (is-a?/c editor<%>)
  keymap : (is-a?/c keymap:aug-keymap<%>)
```

Removes *keymap* from the keymaps chained to *editor*. Also (indirectly) removes all keymaps chained to *keymap* from *editor*, since they are removed when unchaining *keymap* itself.

Each of the keymaps chained to *editor* must be an `keymap:aug-keymap%` and *keymap* cannot be the result of `(send editor get-keymap)` That is, *keymap* must be chained to some keymap attached to the editor.

18 Menu

`menu:can-restore<%>` : `interface?`
  implements: `selectable-menu-item<%>`

Classes created with this mixin remember their keybindings so the keybindings can be removed and then restored.

`(send` *a-menu:can-restore* `restore-keybinding)` → `void`

     Sets the keyboard shortcut to the setting it had when the class was created.

`menu:can-restore-mixin` : `(class? . -> . class?)`
  argument extends/implements: `selectable-menu-item<%>`
  result implements: `menu:can-restore<%>`

`menu:can-restore-underscore<%>` : `interface?`
  implements: `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.

`(send` *a-menu:can-restore-underscore* `erase-underscores)` → `void`

     Erases the underscores in the label of this menu, but remembers them so they can be restores with `restore-underscores`.

`(send` *a-menu:can-restore-underscore* `restore-underscores)`
 → `void`

     Restores underscores in the menu's label to their original state.

`menu:can-restore-underscore-mixin` : `(class? . -> . class?)`
  argument extends/implements: `labelled-menu-item<%>`
  result implements: `menu:can-restore-underscore<%>`

`menu:can-restore-menu-item%` : class?

    superclass: `(menu:can-restore-mixin menu-item%)`

`menu:can-restore-checkable-menu-item%` : class?

    superclass: `(menu:can-restore-mixin checkable-menu-item%)`

`menu:can-restore-underscore-menu%` : class?

    superclass: `(menu:can-restore-underscore-mixin menu%)`

19   Mode

`mode:surrogate-text<%>` : `interface?`

(`send` *a-mode:surrogate-text* `on-enable-surrogate`) → `any`

>   Called by `set-surrogate` to notify the surrogate that it has just become active.

(`send` *a-mode:surrogate-text* `on-disable-surrogate`) → `any`

>   Called by `set-surrogate` to notify the surrogate that it has just been disabled.

`mode:surrogate-text%` : `class?`
  superclass: `object%`
  extends: `mode:surrogate-text<%>`

(`send` *a-mode:surrogate-text* `on-change` *orig*
                                        call-inner) → `any`
  orig : `(is-a?/ctext%)`
  call-inner : `(->any)`

>   Returns the result of invoking `call-super`.

(`send` *a-mode:surrogate-text* `on-char` *orig*
                                        call-super
                                        event)   → `any`
  orig : `(is-a?/ctext%)`
  call-super : `(->any)`
  event : `any/c`

>   Returns the result of invoking `call-super`.

(`send` *a-mode:surrogate-text* `on-default-char` *orig*
                                        call-super
                                        event)   → `any`
  orig : `(is-a?/ctext%)`
  call-super : `(->any)`
  event : `any/c`

>   Returns the result of invoking `call-super`.

(`send` *a-mode:surrogate-text* `on-default-event` *orig*
                                        call-super
                                        event)   → `any`

```
orig : (is-a?/ctext%)
call-super : (->any)
event : any/c
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-display-size orig
                                            call-inner) → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-edit-sequence orig
                                            call-inner) → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-event orig
                                     call-super
                                     event)      → any
  orig : (is-a?/ctext%)
  call-super : (->any)
  event : any/c
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-focus orig
                                     call-super
                                     on?)        → any
  orig : (is-a?/ctext%)
  call-super : (->any)
  on? : any/c
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-load-file orig
                                         call-inner
                                         filename
                                         format)     → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
  filename : any/c
  format : any/c
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-local-char orig
                                          call-super
                                          event)        → any
  orig : (is-a?/ctext%)
  call-super : (->any)
  event : any/c
```

Returns the result of invoking *call-super*.

```
(send a-mode:surrogate-text on-local-event orig
                                           call-super
                                           event)        → any
  orig : (is-a?/ctext%)
  call-super : (->any)
  event : any/c
```

Returns the result of invoking *call-super*.

```
(send a-mode:surrogate-text on-new-box orig
                                       call-super
                                       type)        → any
  orig : (is-a?/ctext%)
  call-super : (->any)
  type : any/c
```

Returns the result of invoking *call-super*.

```
(send a-mode:surrogate-text on-new-image-snip orig
                                              call-super
                                              filename
                                              kind
                                              relative-path?
                                              inline?)
 → any
  orig : (is-a?/ctext%)
  call-super : (->any)
  filename : any/c
  kind : any/c
  relative-path? : any/c
  inline? : any/c
```

Returns the result of invoking *call-super*.

```
(send a-mode:surrogate-text on-paint orig
                                     call-super
                                     before?
                                     dc
                                     left
                                     top
                                     right
                                     bottom
                                     dx
                                     dy
                                     draw-caret)  →  any
  orig : (is-a?/ctext%)
  call-super : (->any)
  before? : any/c
  dc : any/c
  left : any/c
  top : any/c
  right : any/c
  bottom : any/c
  dx : any/c
  dy : any/c
  draw-caret : any/c
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-save-file orig
                                         call-inner
                                         filename
                                         format)     →  any
  orig : (is-a?/ctext%)
  call-inner : (->any)
  filename : any/c
  format : any/c
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-snip-modified orig
                                             call-inner
                                             snip
                                             modified?)  →  any
  orig : (is-a?/ctext%)
  call-inner : (->any)
  snip : any/c
  modified? : any/c
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-change-style orig
                                            call-inner
                                            start
                                            len)         → any
```
  orig : (is-a?/ctext%)
  call-inner : (->any)
  start : any/c
  len : any/c

    Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-delete orig
                                      call-inner
                                      start
                                      len)         → any
```
  orig : (is-a?/ctext%)
  call-inner : (->any)
  start : any/c
  len : any/c

    Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-insert orig
                                      call-inner
                                      start
                                      len)         → any
```
  orig : (is-a?/ctext%)
  call-inner : (->any)
  start : any/c
  len : any/c

    Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text on-new-string-snip orig
                                               call-super)
 → any
```
  orig : (is-a?/ctext%)
  call-super : (->any)

    Returns the result of invoking *call-super*.

```
(send a-mode:surrogate-text on-new-tab-snip orig
                                            call-super) → any
```
  orig : (is-a?/ctext%)
  call-super : (->any)

    Returns the result of invoking *call-super*.

```
(send a-mode:surrogate-text on-set-size-constraint orig
                                                    call-inner)
 → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
```
Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text after-change-style orig
                                                call-inner
                                                start
                                                len)
 → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
  start : any/c
  len : any/c
```
Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text after-delete orig
                                         call-inner
                                         start
                                         len)          → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
  start : any/c
  len : any/c
```
Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text after-insert orig
                                         call-inner
                                         start
                                         len)          → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
  start : any/c
  len : any/c
```
Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text after-set-position orig
                                               call-inner)
 → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text after-set-size-constraint
 orig
 call-inner)
→ any
 orig : (is-a?/ctext%)
 call-inner : (->any)
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text after-edit-sequence orig
                                                call-inner)
→ any
 orig : (is-a?/ctext%)
 call-inner : (->any)
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text after-load-file orig
                                            call-inner
                                            success?)   → any
 orig : (is-a?/ctext%)
 call-inner : (->any)
 success? : any/c
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text after-save-file orig
                                            call-inner
                                            success?)   → any
 orig : (is-a?/ctext%)
 call-inner : (->any)
 success? : any/c
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text can-change-style? orig
                                              call-inner
                                              start
                                              len)       → any
 orig : (is-a?/ctext%)
 call-inner : (->any)
 start : any/c
 len : any/c
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text can-delete? orig
                                         call-inner
                                         start
                                         len)         → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
  start : any/c
  len : any/c
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text can-insert? orig
                                         call-inner
                                         start
                                         len)         → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
  start : any/c
  len : any/c
```

Returns the result of invoking `call-super`.

```
 (send a-mode:surrogate-text can-set-size-constraint?
  orig
  call-inner)
→ any
  orig : (is-a?/ctext%)
  call-inner : (->any)
```

Returns the result of invoking `call-super`.

```
(send a-mode:surrogate-text can-do-edit-operation? orig
                                                    call-super
                                                    op)
 → any
  orig : (is-a?/ctext%)
  call-super : (->any)
  op : any/c
(send a-mode:surrogate-text can-do-edit-operation? orig
                                                    call-super
                                                    op
                                                    recursive?)
 → any
  orig : (is-a?/ctext%)
  call-super : (->any)
  op : any/c
```

131

recursive? : any/c

Returns the result of invoking `call-super`.

(send *a-mode:surrogate-text* `can-load-file?` *orig*
                                          call-inner
                                          filename
                                          format)   → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
  filename : any/c
  format : any/c

Returns the result of invoking `call-super`.

(send *a-mode:surrogate-text* `can-save-file?` *orig*
                                          call-inner
                                          filename
                                          format)   → any
  orig : (is-a?/ctext%)
  call-inner : (->any)
  filename : any/c
  format : any/c

Returns the result of invoking `call-super`.

(send *a-mode:surrogate-text* `put-file` *orig*
                                     call-super
                                     directory
                                     default-name) → any
  orig : (is-a?/ctext%)
  call-super : (->any)
  directory : any/c
  default-name : any/c

Returns the result of invoking `call-super`.

`mode:host-text<%>` : `interface?`

(send *a-mode:host-text* `get-surrogate`)
 → (or/c false/c (is-a?/c mode:surrogate-text<%>))

Returns the currently active surrogate.

(send *a-mode:host-text* `set-surrogate` *surrogate*) → void?
  surrogate : (or/c false/c (is-a?/c mode:surrogate-text<%>))

Sets the current surrogate to *surrogate*.

`mode:host-text-mixin` : `(class? . -> . class?)`
  result implements: `mode:host-text<%>`

`(send ` *a-mode:host-text* ` on-change)` $\rightarrow$ `any`

    Delegates to the result of `get-surrogate` if it is not `#f`.

`(send ` *a-mode:host-text* ` on-char ` *event*`)` $\rightarrow$ `any`
  event : `any/c`

    Delegates to the result of `get-surrogate` if it is not `#f`.

`(send ` *a-mode:host-text* ` on-default-char ` *event*`)` $\rightarrow$ `any`
  event : `any/c`

    Delegates to the result of `get-surrogate` if it is not `#f`.

`(send ` *a-mode:host-text* ` on-default-event ` *event*`)` $\rightarrow$ `any`
  event : `any/c`

    Delegates to the result of `get-surrogate` if it is not `#f`.

`(send ` *a-mode:host-text* ` on-display-size)` $\rightarrow$ `any`

    Delegates to the result of `get-surrogate` if it is not `#f`.

`(send ` *a-mode:host-text* ` on-edit-sequence)` $\rightarrow$ `any`

    Delegates to the result of `get-surrogate` if it is not `#f`.

`(send ` *a-mode:host-text* ` on-event ` *event*`)` $\rightarrow$ `any`
  event : `any/c`

    Delegates to the result of `get-surrogate` if it is not `#f`.

`(send ` *a-mode:host-text* ` on-focus ` *on?*`)` $\rightarrow$ `any`
  on? : `any/c`

    Delegates to the result of `get-surrogate` if it is not `#f`.

`(send ` *a-mode:host-text* ` on-load-file ` *filename*
                                format`)` $\rightarrow$ `any`
  filename : `any/c`
  format : `any/c`

Delegates to the result of `get-surrogate` if it is not `#f`.

`(send a-mode:host-text on-local-char event)` → any
  `event` : any/c

Delegates to the result of `get-surrogate` if it is not `#f`.

`(send a-mode:host-text on-local-event event)` → any
  `event` : any/c

Delegates to the result of `get-surrogate` if it is not `#f`.

`(send a-mode:host-text on-new-box type)` → any
  `type` : any/c

Delegates to the result of `get-surrogate` if it is not `#f`.

`(send a-mode:host-text on-new-image-snip filename`
                                         `kind`
                                         `relative-path?`
                                         `inline?)`   → any
  `filename` : any/c
  `kind` : any/c
  `relative-path?` : any/c
  `inline?` : any/c

Delegates to the result of `get-surrogate` if it is not `#f`.

`(send a-mode:host-text on-paint before?`
                                 `dc`
                                 `left`
                                 `top`
                                 `right`
                                 `bottom`
                                 `dx`
                                 `dy`
                                 `draw-caret)` → any
  `before?` : any/c
  `dc` : any/c
  `left` : any/c
  `top` : any/c
  `right` : any/c
  `bottom` : any/c
  `dx` : any/c
  `dy` : any/c
  `draw-caret` : any/c

Delegates to the result of `get-surrogate` if it is not `#f`.

```
(send a-mode:host-text on-save-file filename
                                    format)  → any
  filename : any/c
  format : any/c
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text on-snip-modified snip
                                        modified?) → any
  snip : any/c
  modified? : any/c
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text on-change-style start
                                        len)  → any
  start : any/c
  len : any/c
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text on-delete start len) → any
  start : any/c
  len : any/c
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text on-insert start len) → any
  start : any/c
  len : any/c
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text on-new-string-snip) → any
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text on-new-tab-snip) → any
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text on-set-size-constraint) → any
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text after-change-style start
                                           len)  → any
  start : any/c
```

len : any/c

    Delegates to the result of get-surrogate if it is not #f.

(send *a-mode:host-text* after-delete *start*
                                   len) → any

start : any/c

len : any/c

    Delegates to the result of get-surrogate if it is not #f.

(send *a-mode:host-text* after-insert *start*
                                   len) → any

start : any/c

len : any/c

    Delegates to the result of get-surrogate if it is not #f.

(send *a-mode:host-text* after-set-position) → any

    Delegates to the result of get-surrogate if it is not #f.

(send *a-mode:host-text* after-set-size-constraint) → any

    Delegates to the result of get-surrogate if it is not #f.

(send *a-mode:host-text* after-edit-sequence) → any

    Delegates to the result of get-surrogate if it is not #f.

(send *a-mode:host-text* after-load-file *success?*) → any
  success? : any/c

    Delegates to the result of get-surrogate if it is not #f.

(send *a-mode:host-text* after-save-file *success?*) → any
  success? : any/c

    Delegates to the result of get-surrogate if it is not #f.

(send *a-mode:host-text* can-change-style? *start*
                                       len) → any

start : any/c

len : any/c

    Delegates to the result of get-surrogate if it is not #f.

(send *a-mode:host-text* can-delete? *start*
                                len) → any

start : any/c

```
len : any/c
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text can-insert? start
                                   len)  → any
  start : any/c
  len : any/c
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text can-set-size-constraint?) → any
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text can-do-edit-operation? op) → any
  op : any/c
(send a-mode:host-text can-do-edit-operation? op
                                              recursive?) → any
  op : any/c
  recursive? : any/c
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text can-load-file? filename
                                      format)  → any
  filename : any/c
  format : any/c
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text can-save-file? filename
                                      format)  → any
  filename : any/c
  format : any/c
```

Delegates to the result of get-surrogate if it is not #f.

```
(send a-mode:host-text put-file directory
                                default-name) → any
  directory : any/c
  default-name : any/c
```

Delegates to the result of get-surrogate if it is not #f.

20 Number Snip

`number-snip:snip-class%` : `class?`

  superclass: `snip-class%`

`(send` *a-number-snip:snip-class* `read` *f*`)` → `snip`

  f : `stream-in`

     Overrides `read` in `snip-class%`.

     Constructs a number snip from its input.

`(number-snip:make-repeating-decimal-snip` *num*
                                       show-prefix?`)`
→ `(is-a?/c` `snip%)`

  num : `number?`

  show-prefix? : `boolean?`

Makes a number snip that shows the decimal expansion for `number` The boolean indicates if a `#e` prefix appears on the number.

See also `number-snip:make-fraction-snip`.

`(number-snip:make-fraction-snip` *num*
                               show-prefix-in-decimal-view?`)`
→ `(is-a?/c` `snip%)`

  num : `number?`

  show-prefix-in-decimal-view? : `boolean?`

Makes a number snip that shows a fractional view of `number`. The boolean indicates if a `#e` prefix appears on the number, when shown in the decimal state.

See also `number-snip:make-repeating-decimal-snip`.

21 Panel

`panel:single<%>` : `interface?`
  implements: `area-container<%>`

See `panel:single-mixin%`.

`(send` *a-panel:single* `active-child` *child*`)` → `void`
  child : `(is-a?/c area<%>)`
`(send` *a-panel:single* `active-child)` → `(is-a?/c area<%>)`

> Sets the active child to be *child*

> Returns the current active child.

`panel:single-mixin` : `(class? . -> . class?)`
  argument extends/implements: `area-container<%>`
  result 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.

`(send` *a-panel:single* `after-new-child` *child*`)` → `void`
  child : `subarea<%>`

> Overrides `after-new-child` in `area-container<%>`.

> Hides this child by calling `(send` *child* `show #f)`, unless this is the first child in which case it does nothing.

`(send` *a-panel:single* `container-size)`
 → `exact-integer exact-integer`

> Overrides `container-size` in `area-container<%>`.

> Returns the maximum width of all the children and the maximum height of all of the children.

`(send` *a-panel:single* `place-children)`
 → `(listof (list exact-integer exact-integer exact-integer exact-integer))`

Overrides `place-children` in `area-container<%>`.

Returns the positions for single panels and panes.

`panel:single-window<%>` : `interface?`
  implements: `panel:single<%>`
              `window<%>`

`panel:single-window-mixin` : `(class? . -> . class?)`
  argument extends/implements: `panel:single<%>`
                               `window<%>`
  result implements: `panel:single-window<%>`

`(send a-panel:single-window container-size info)`
`→ exact-integer exact-integer`
  `info` : `(list-of (list exact-integer exact-integer boolean boolean?))`

    Overrides `container-size` in `area-container<%>`.

    Factors the border width into the size calculation.

`panel:single%` : `class?`
  superclass: `(panel:single-window-mixin (panel:single-mixin panel%))`

`panel:single-pane%` : `class?`
  superclass: `(panel:single-mixin pane%)`

`panel:dragable<%>` : `interface?`
  implements: `window<%>`
              `area-container<%>`

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 and dragging the empty space between the children.

(send *a-panel:dragable* `after-percentage-change`) → `void`

> 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.
>
> Use `get-percentages` to find the current percentages.

(send *a-panel:dragable* `set-percentages` *new-percentages*) → `void`
  new-percentages : (`listof` `number`)

> Call this method to set the percentages that each window takes up of the panel.
>
> 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`.

(send *a-panel:dragable* `get-percentages`) → (`listof` `numbers`)

> Return the current percentages of the children.

(send *a-panel:dragable* `get-vertical?`) → `boolean?`

> 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.

`panel:vertical-dragable<%>` : `interface?`
  implements: `panel:dragable<%>`

A panel that implements `panel:vertical-dragable<%>`. It aligns its children vertically.

`panel:horizontal-dragable<%>` : `interface?`
  implements: `panel:dragable<%>`

A panel that implements `panel:horizontal-dragable<%>`. It aligns its children horizontally.

`panel:dragable-mixin` : `(class? . -> . class?)`
  argument extends/implements: `window<%>`
  `area-container<%>`
  result implements: `panel:dragable<%>`

This mixin adds the `panel:dragable<%>` functionality to a `panel%`.

It is not useful unless the `get-vertical?` method is overridden.

`(send a-panel:dragable after-new-child child)` → `void`
  `child` : `(instance-of (is-a?/c area<%>))`

> Overrides `after-new-child` in `area-container<%>`.

> Updates the number of percentages to make sure that it matches the number of children and calls `after-percentage-change`.

`(send a-panel:dragable on-subwindow-event receiver`
  `event)` → `boolean?`
  `receiver` : `(instanceof window<%>)`
  `event` : `(instanceof mouse-event%)`

> Overrides `on-subwindow-event` in `window<%>`.

> When the cursor is dragging the middle bar around, this method handles the resizing of the two panes.

`(send a-panel:dragable 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`

> Overrides `place-children` in `area-container<%>`.

> 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.

`(send a-panel:dragable container-size info)` → `two`
  `info` : `list`

> Overrides `container-size` in `area-container<%>`.

> Computes the minimum size the panel would have to be in order to have the current percentages (see `get-percentages`).

`panel:vertical-dragable-mixin` : `(class? . -> . class?)`
   argument extends/implements: `panel:dragable<%>`
   result implements: `panel:vertical-dragable<%>`

This mixin merely overrides the `get-vertical?` method of the `panel:dragable-mixin`
to return `#t`.

`(send `*a-panel:vertical-dragable*` get-vertical?)` → `boolean?`

    Overrides `get-vertical?` in `panel:dragable<%>`.

    Returns `#t`.

`panel:horizontal-dragable-mixin` : `(class? . -> . class?)`
   argument extends/implements: `panel:dragable<%>`
   result implements: `panel:vertical-dragable<%>`

This mixin merely overrides the `get-vertical?` method of the `panel:dragable-mixin`
to return `#f`.

`(send `*a-panel:horizontal-dragable*` get-vertical?)` → `boolean?`

    Overrides `get-vertical?` in `panel:dragable<%>`.

    Returns `#f`.

`panel:vertical-dragable%` : `class?`
   superclass: `(panel:vertical-dragable-mixin (panel:dragable-mixin vertical-panel%))`

`panel:horizontal-dragable%` : `class?`
   superclass: `(panel:horizontal-dragable-mixin (panel:dragable-mixin horizontal-panel%))`

22 Pasteboard

`pasteboard:basic%` : class?
  superclass: `(editor:basic-mixin pasteboard%)`

`pasteboard:standard-style-list%` : class?
  superclass: `(editor:standard-style-list-mixin pasteboard:basic%)`

`pasteboard:keymap%` : class?
  superclass: `(editor:keymap-mixin pasteboard:standard-style-list%)`

`pasteboard:file%` : class?
  superclass: `(editor:file-mixin pasteboard:keymap%)`

`pasteboard:backup-autosave%` : class?
  superclass: `(editor:backup-autosave-mixin pasteboard:file%)`

`pasteboard:info%` : class?
  superclass: `(editor:info-mixin pasteboard:backup-autosave%)`

23   Path Utils

(`path-utils:generate-autosave-name` *filename*) → string?
  filename : string?

Generates a name for an autosave file from *filename*.

(`path-utils:generate-backup-name` *filename*) → path?
  filename : path?

Generates a name for an backup file from *filename*.

24 Preferences

```
(preferences:put-preferences/gui name-list
                                 val-list) → any
  name-list : (listof symbol?)
  val-list : (listof any/c)
```

Like `put-preferences`, but has more sophisticated error handling. In particular, it

- waits for three consecutive failures before informing the user

- gives the user the opportunity to "steal" the lockfile after the third failure, and

- when failures occur, it remembers what its arguments were and if any preference save eventually succeeds, all of the past failures are also written at that point.

```
(preferences:add-panel labels f) → void?
  labels : (or/c string? (cons/c string? (listof string?)))
  f : (->d ([parent (is-a?/c area-container-window<%>)])
           ()
           [_
            (let ([old-children (send parent get-children)])
              (and/c (is-a?/c area-container-window<%>)
                     (λ (child)
                       (andmap eq?
                               (append old-children (list child))
                               (send parent get-children)))))])
```

`preferences:add-preference-panel` adds the result of `f` with name `labels` to the preferences dialog box.

The labels determine where this preference panel is placed in the dialog. If the list is just one string, the preferences panel is placed at the top level of the dialog. If there are more strings, a hierarchy of nested panels is created and the new panel is added at the end. If multiple calls to `preferences:add-preference-panel` pass the same prefix of strings, those panels are placed in the same children.

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-editor-checkbox-panel) → void?
```

Adds a preferences panel for configuring options related to editing.

`(preferences:add-general-checkbox-panel)` $\rightarrow$ `void?`

Adds a catch-all preferences panel for options.

`(preferences:add-warnings-checkbox-panel)` $\rightarrow$ `void?`

Adds a preferences panel for configuring options relating to warnings.

`(preferences:add-scheme-checkbox-panel)` $\rightarrow$ `void?`

Adds a preferences panel for configuring options related to Racket.

```
(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-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 Racket preferences panel to add new children to the 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 editor preferences panel to add new children to the panel.

```
(preferences:add-to-general-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 general preferences panel to add new children to the panel.

`(preferences:add-font-panel)` $\rightarrow$ `void?`

Adds a font selection preferences panel to the preferences dialog.

`(preferences:show-dialog)` → `void?`

Shows the preferences dialog.

`(preferences:hide-dialog)` → `void?`

Hides the preferences dialog.

`(preferences:add-on-close-dialog-callback cb)` → `void?`
  `cb` : `(-> void?)`

Registers *cb*. Next time the user clicks the OK button the preferences dialog, all of the *cb* functions are called, assuming that each of the callbacks passed to `preferences:add-can-close-dialog-callback` succeed.

`(preferences:add-can-close-dialog-callback cb)` → `void?`
  `cb` : `(-> boolean?)`

Registers *cb*. Next time the user clicks the OK button the preferences dialog, all of the *cb* functions are called. If any of them return `#f`, the dialog is not closed.

See also `preferences:add-on-close-dialog-callback`.

25 Preferences, Textual

```
(require framework/preferences)
```

```
(preferences:get symbol) → any/c
  symbol : symbol?
```

See also `preferences:set-default`.

`preferences:get` returns the value for the preference `symbol`. It raises `exn:unknown-preference` if the preference's default has not been set.

```
(preferences:set symbol value) → void?
  symbol : symbol?
  value : any/c
```

See also `preferences:set-default`.

`preferences:set-preference` sets the preference `symbol` to `value`. This should be called when the users requests a change to a preference.

This function immediately writes the preference value to disk.

It raises `exn:unknown-preference` if the preference's default has not been set.

```
(preferences:add-callback p f [weak?]) → (-> void?)
  p : symbol?
  f : (λ (x) (and (procedure? x) (procedure-arity-includes? x 2)))
  weak? : boolean? = #f
```

This function adds a callback which is called with a symbol naming a preference and its value, when the preference changes. `preferences:add-callback` returns a thunk, which when invoked, removes the callback from this preference.

If `weak?` is true, the preferences system will only hold on to the callback weakly.

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 exn:unknown-preference` if the preference has not been set.

```
(preferences:set-default  symbol
                          value
                          test
                          [#:aliases aliases
                           #:rewrite-aliases rewrite-aliases])
 → void?
  symbol : symbol?
  value : any/c
  test : (any/c . -> . any)
  aliases : (listof symbol?) = '()
  rewrite-aliases :  (listof (-> any/c any))
                  = (map (lambda (x) (values)) aliases)
```

This function must be called every time your application starts up, before any call to `preferences:get` or `preferences:set` (for any given preference).

If you use `preferences:set-un/marshall`, you must call this function before calling it.

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 *test* argument 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.

The *aliases* and *rewrite-aliases* arguments aids in renaming preferences. If *aliases* is present, it is expected to be a list of symbols that correspond to old versions of the preferences. It defaults to `'()`. If *rewrite-aliases* is present, it is used to adjust the old values of the preferences when they are present in the saved file.

```
(preferences:set-un/marshall symbol
                             marshall
                             unmarshall) → void?
  symbol : symbol?
  marshall : (any/c . -> . printable/c)
  unmarshall : (printable/c . -> . any/c)
```

`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 its 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.

The `marshall` function might be called with any value returned from `read` and it must not raise an error (although it can return arbitrary results if it gets bad input). This might happen when the preferences file becomes corrupted, or is edited by hand.

`preference:set-un/marshall` must be called before calling `preferences:get`,`preferences:set`.

(`preferences:restore-defaults`) → `void?`

(`preferences:restore-defaults`) restores the users' configuration to the default preferences.

```
(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) → boolean?
  exn : any/c
```

Determines if a value is an unknown preference exn.

```
(preferences:low-level-put-preferences)
 → ((listof symbol?) (listof any/c) . -> . any)
(preferences:low-level-put-preferences put-preferences) → void?
  put-preferences : ((listof symbol?) (listof any/c) . -> . any)
```

This parameter's value is called to save preference the preferences file. Its interface should be just like mzlib's *put-preferences*.

```
(preferences:low-level-get-preference)
 → (->* [symbol?] [(-> any)] any)
(preferences:low-level-get-preference get-preference) → void?
  get-preference : (->* [symbol?] [(-> any)] any)
```

This parameter's value is called to get a preference from the preferences file. Its interface should be just like mzlib's *get-preference*.

```
(preferences:snapshot? arg) → boolean?
  arg : any/c
```

Determines if its argument is a preferences snapshot.

See also `preferences:get-prefs-snapshot` and `preferences:restore-prefs-snapshot`.

```
(preferences:restore-prefs-snapshot snapshot) → void?
  snapshot : preferences:snapshot?
```

Restores the preferences saved in *snapshot*.

See also `preferences:get-prefs-snapshot`.

```
(preferences:get-prefs-snapshot) → preferences:snapshot?
```

Caches all of the current values of the preferences and returns them. For any preference that has marshalling and unmarshalling set (see `preferences:set-un/marshall`), the preference value is copied by passing it through the marshalling and unmarshalling process. Other values are not copied, but references to them are instead saved.

See also `preferences:restore-prefs-snapshot`.

26 Racket

`scheme:sexp-snip<%>` : `interface?`

`(send` *a-scheme:sexp-snip* `get-saved-snips)` → `(listof snip%)`

> This returns the list of snips hidden by the sexp snip.

`scheme:sexp-snip%` : `class?`
  superclass: `snip%`
  extends: `scheme:sexp-snip<%>`
            `readable-snip<%>`

`(send` *a-scheme:sexp-snip* `get-text` *offset*
                                         num
                                         [*flattened?*])` → `string`
  offset : `number`
  num : `number`
  flattened? : `boolean` = `|#f|`

> Overrides `get-text` in `snip%`.

> Returns the concatenation of the text for all of the hidden snips.

`(send` *a-scheme:sexp-snip* `copy)` → `(is-a?/c scheme:sexp-snip%)`

> Overrides `copy` in `snip%`.

> Returns a copy of this snip that includes the hidden snips.

`(send` *a-scheme:sexp-snip* `write` *stream-out*`)` → `void`
  stream-out : `editor-stream-out%`

> Overrides `write` in `snip%`.

> Saves the embedded snips

```
(send a-scheme:sexp-snip draw dc
                              x
                              y
                              left
                              top
                              right
                              bottom
                              dx
                              dy
                              draw-caret) → void
  dc : dc<%>
  x : real
  y : real
  left : real
  top : real
  right : real
  bottom : real
  dx : real
  dy : real
  draw-caret : symbol?
```

Overrides draw in snip%.

Draws brackets with a centered ellipses between them.

```
(send a-scheme:sexp-snip get-extent dc
                                    x
                                    y
                                   [w
                                    h
                                    descent
                                    space
                                    lspace
                                    rspace]) → void
  dc : (is-a?/c dc<%>)
  x : real
  y : real
  w : boxed = |#f|
  h : boxed = |#f|
  descent : boxed = |#f|
  space : boxed = |#f|
  lspace : boxed = |#f|
  rspace : boxed = |#f|
```

Overrides get-extent in snip%.

Returns a size corresponding to what this snip draws.

```
scheme:text<%> : interface?
  implements:  text:basic<%>
               mode:host-text<%>
               color:text<%>
```

Texts matching this interface support Racket mode operations.

```
(send a-scheme:text get-limit start) → int
  start : exact-integer
```

> Returns a limit for backward-matching parenthesis starting at position *start*.

```
(send a-scheme:text balance-parens key-event) → void
  key-event : (instance key-event%)
```

> 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 tabify-on-return?) → boolean?
```

> The result of this method is used to determine if the return key automatically tabs over to the correct position.

> Override it to change its behavior.

```
(send a-scheme:text tabify [start-pos]) → void
  start-pos :  exact-integer
            = (send this text get-start-position)
```

> Tabs the line containing by *start-pos*

```
(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*.

```
(send a-scheme:text tabify-all) → void
```

> Tabs all lines.

```
(send a-scheme:text insert-return) → void
```

155

Inserts a newline into the buffer. If `tabify-on-return?` returns `#t`, this will tabify the new line.

```
(send a-scheme:text box-comment-out-selection start-pos
                                              end-pos)  → void
  start-pos : (or/c (symbols 'start) exact-integer?)
  end-pos : (or/c (symbols 'end) exact-integer?)
```

This method comments out a selection in the text by putting it into a comment box.

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.

```
(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.

```
(send a-scheme:text uncomment-selection start
                                        end)  → void
  start : int
  end : int
```

Uncomments the lines containing positions `start` through `end`.

```
(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.

```
(send a-scheme:text remove-sexp start) → void
  start : exact-integer
```

Forward-deletes the S-expression starting after the position `start`.

```
(send a-scheme:text forward-sexp start) → exact-integer
  start : |#t|
```

Moves forward over the S-expression starting at position `start`.

```
(send a-scheme:text flash-forward-sexp start-pos) → void
```

```
start-pos : exact-integer
```
Flashes the parenthesis that closes the sexpression at `start-pos`.

```
(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.

```
(send a-scheme:text flash-backward-sexp start-pos) → void
  start-pos : exact-integer
```
Flashes the parenthesis that opens the sexpression at `start-pos`.

```
(send a-scheme:text backward-sexp start-pos) → void
  start-pos : exact-integer
```
Move the caret backwards one sexpression

Moves the caret to the beginning of the sexpression that ends at `start-pos`.

```
(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`.

```
(send a-scheme:text up-sexp start) → void
  start : exact-integer
```
Moves backward out of the S-expression containing the position `start`.

```
(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`.

```
(send a-scheme:text down-sexp start) → void
  start : exact-integer
```
Moves forward into the next S-expression after the position `start`.

```
(send a-scheme:text remove-parens-forward start) → void
```

start : exact-integer

> Removes the parentheses from the S-expression starting after the position *start*.

(send *a-scheme:text* select-forward-sexp *start*) → |#t|
  start : exact-integer

> Selects the next S-expression, starting at position *start*.

(send *a-scheme:text* select-backward-sexp *start*) → |#t|
  start : exact-integer

> Selects the previous S-expression, starting at position *start*.

(send *a-scheme:text* select-up-sexp *start*) → |#t|
  start : exact-integer

> Selects the region to the enclosing S-expression, starting at position *start*.

(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*.

(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*.

(send *a-scheme:text* mark-matching-parenthesis *pos*) → void
  pos : exact-positive-integer

> If the paren after *pos* is matched, this method highlights it and its matching counterpart in dark green.

(send *a-scheme:text* get-tab-size) → exact-integer

> This method returns the current size of the tabs for scheme mode. See also set-tab-size.

(send *a-scheme:text* set-tab-size *new-size*) → void
  new-size : exact-integer

> This method sets the tab size for this text.

(send *a-scheme:text* introduce-let-ans) → void

> Adds a let around the current s-expression and a printf into the body of the let.

(send *a-scheme:text* move-sexp-out) → void

> Replaces the sexpression surrounding the insertion point with the sexpression following the insertion point.

---

scheme:text-mixin : (class? . -> . class?)
  argument extends/implements: text:basic<%>
                                mode:host-text<%>
                                color:text<%>
                                text:autocomplete<%>
  result implements: scheme:text<%>

This mixin adds functionality for editing Racket files.

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

(send *a-scheme:text* get-word-at *pos*) → string
  pos : positive-exact-integer

> Overrides get-word-at in text:autocomplete<%>.

> Returns the word just before *pos*, which is then used as the prefix for auto-completion.

---

scheme:text-mode<%> : interface?

The result of scheme:text-mode-mixin implements this interface.

---

scheme:text-mode-mixin : (class? . -> . class?)
  argument extends/implements: color:text-mode<%>
                                mode:surrogate-text<%>
  result implements: scheme:text-mode<%>

This mixin adds Racket mode functionality to the mode that it is mixed into. The resulting mode assumes that it is only set to an editor that is the result of scheme:text-mixin.

(send *a-scheme:text-mode* on-disable-surrogate) → void

> Overrides on-disable-surrogate in mode:surrogate-text<%>.

159

Removes the scheme keymap (see also `scheme:get-keymap`) and disables any parenthesis highlighting in the host editor.

(send `a-scheme:text-mode` `on-enable-surrogate`) → `void`

Overrides `on-enable-surrogate` in `mode:surrogate-text<%>`.

Adds the scheme keymap (see also `scheme:get-keymap`) and enables a parenthesis highlighting in the host editor.

`scheme:set-mode-mixin` : (`class?` . `->` . `class?`)
  argument extends/implements: `scheme:text<%>`
                               `mode:host-text<%>`

This mixin creates a new instance of `scheme:text-mode%` and installs it, by calling its own `set-surrogate` method with the object.

`scheme:text%` : `class?`
  superclass: (`scheme:set-mode-mixin` (`scheme:text-mixin` (`text:autocomplete-mixin` (`mode:host-text`

`scheme:text-mode%` : `class?`
  superclass: (`scheme:text-mode-mixin` `color:text-mode%`)

(`scheme:text-balanced?` `text` [`start` `end`]) → `boolean?`
  `text` : (`is-a?/c` `text%`)
  `start` : `number?` = 0
  `end` : (`or/c` `false/c` `number?`) = `#f`

Determines if the range in the editor from `start` to `end` in `text` has at least one complete s-expression and there are no incomplete s-expressions. If `end` is `#f`, it defaults to the last position of the `text`.

The implementation of this function creates a port with `open-input-text-editor` and then uses 'read' to parse the range of the buffer.

(`scheme:add-preferences-panel`) → `void?`

160

Adds a tabbing preferences panel to the preferences dialog.

`(scheme:get-keymap)` → `(is-a?/c keymap%)`

Returns a keymap with binding suitable for Racket.

`(scheme:add-coloring-preferences-panel)` → `any`

Installs the "Racket" preferences panel in the "Syntax Coloring" section.

`(scheme:get-color-prefs-table)`
 → `(listof (list/c symbol? (is-a?/c color%)))`

Returns a table mapping from symbols (naming the categories that the online colorer uses for Racket mode coloring) to their colors.

These symbols are suitable for input to `scheme:short-sym->pref-name` and `scheme:short-sym->style-name`.

See also `scheme:get-white-on-black-color-prefs-table`.

`(scheme:get-white-on-black-color-prefs-table)`
 → `(listof (list/c symbol? (is-a?/c color%)))`

Returns a table mapping from symbols (naming the categories that the online colorer uses for Racket mode coloring) to their colors when the user chooses the white-on-black mode in the preferences dialog.

See also `scheme:get-color-prefs-table`.

`(scheme:short-sym->pref-name` *short-sym*`)` → `symbol?`
  short-sym : `symbol?`

Builds the symbol naming the preference from one of the symbols in the table returned by `scheme:get-color-prefs-table`.

`(scheme:short-sym->style-name` *short-sym*`)` → `string?`
  short-sym : `symbol?`

Builds the symbol naming the editor style from one of the symbols in the table returned by `scheme:get-color-prefs-table`. This style is a named style in the style list returned by `editor:get-standard-style-list`.

`(scheme:get-wordbreak-map)` → `(is-a?/c editor-wordbreak-map%)`

This method returns a `editor-wordbreak-map%` that is suitable for Racket.

`(scheme:init-wordbreak-map key)` → `void?`
  `key` : `(is-a?/c keymap%)`

Initializes the workdbreak map for `keymap`.

`(scheme:setup-keymap keymap)` → `void?`
  `keymap` : `(is-a?/c keymap%)`

Initializes *keymap* with Racket-mode keybindings.

27 Text

```
text:basic<%> : interface?
  implements: editor:basic<%>
              text%
```

Classes matching this interface are expected to implement the basic functionality needed by the framework.

```
(send a-text:basic highlight-range  start
                                    end
                                    color
                                    [caret-space
                                    priority
                                    style])     → (-> void)
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
  color : (or/c string? (is-a?/c color%))
  caret-space : boolean? = #f
  priority : (symbols 'high 'low) = 'low
  style : (symbols 'rectangle 'ellipse 'hollow-ellipse 'dot)
        = 'rectangle
```

This function highlights a region of text in the buffer.

The range between *start* and *end* will be highlighted with the color in color, if the style is 'rectangle (the default). If the style is 'ellipse, then an ellipse is drawn around the range in the editor, using the color. If the style is 'hollow-ellipse, then the outline of an ellipse is drawn around the range in the editor, using the color.

If the style is 'dot, then *start* and *end* must be the same, and a dot is drawn at the bottom of that position in the editor.

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 some platforms, 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.

See also `unhighlight-range`.

```
(send a-text:basic unhighlight-range  start
                                      end
                                      color
                                     [caret-space
                                      style])      → void?
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
  color : (or/c string? (is-a?/c color%))
  caret-space : boolean? = #f
  style : (symbols 'rectangle 'ellipse 'hollow-ellipse)
        = 'rectangle
```

This method removes the highlight from a region of text in the buffer.

The region must match up to a region specified from an earlier call to `highlight-range`.

```
(send a-text:basic get-highlighted-ranges) → (listof range)
```

Returns a list of (opaque) values representing the active ranges in the editor.

```
(send a-text:basic get-styles-fixed) → boolean?
```

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 `set-styles-fixed`.

```
(send a-text:basic get-fixed-style) → (is-a?/c style<%>)
```

Returns the style used by `set-styles-fixed`when setting the styles.

```
(send a-text:basic set-styles-fixed fixed?) → void
  fixed? : boolean?
```

Sets the styles fixed parameter of this `text%`. See also `get-styles-fixed` and `get-fixed-style`.

```
(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
```

164

This moves or copies text and snips to another edit.

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`.

(send *a-text:basic* initial-autowrap-bitmap)
→ (union |#f| (instance 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`

Defaultly returns the result of `icon:get-autowrap-bitmap`

(send *a-text:basic* get-port-name) → symbol?

The result of this method is a symbol that identifies this editor and that is used as the port-name of a port that is read from this editor if this editor is used in DrRacket. See also `port-name-matches?`.

(send *a-text:basic* port-name-matches? *id*) → boolean?
  id : (or/c path? symbol?)

Indicates if *id* matches the port name of this file. If the file is saved, the port name matches when the save file is the path as *id*. If the file has not been saved, the port name matches if the symbol is the same as the result of `port-name-matches?`.

(send *a-text:basic* get-edition-number)
→ exact-nonnegative-integer?

Returns a number that increments everytime something in the editor changes.

The number is updated in `after-insert` in `text%` and `after-delete` in `text%`.

---

text:basic-mixin : (class? . -> . class?)
  argument extends/implements: editor:basic<%>
                               text%
  result implements: text: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.

```
(send a-text:basic on-paint before?
                              dc
                              left
                              top
                              right
                              bottom
                              dx
                              dy
                              draw-caret) → void
  before? : any/c
  dc : (is-a?/c dc<%>)
  left : real?
  top : real?
  right : real?
  bottom : real?
  dx : real?
  dy : real?
  draw-caret : (one-of/c 'no-caret  'show-inactive-caret  'show-caret)
```

Overrides on-paint in editor<%>.

Draws the rectangles installed by highlight-range.

```
(send a-text:basic on-insert start end) → void
  start : exact-int
  end : exact-int
```

Augments on-insert in text%.

See set-styles-fixed.

```
(send a-text:basic after-insert start len) → void
  start : exact-nonnegative-integer?
  len : exact-nonnegative-integer?
```

Augments after-insert in text%.

See set-styles-fixed.

```
(send a-text:basic put-file directory
                              default-name) → void
  directory : path
  default-name : path
```

Overrides put-file in editor<%>.

Like put-file but uses finder:put-file instead of put-file.

```
text:first-line<%> : interface?
```
  implements: `text%`

Objects implementing this interface, when `highlight-first-line` is invoked with `#t`, always show their first line, even with scrolled (as long as `first-line-currently-drawn-specially?` returns `#t`).

```
(send a-text:first-line highlight-first-line on?) → void?
  on? : boolean?
```
> Call this method to enable special treatment of the first line in the editor.

```
(send a-text:first-line first-line-currently-drawn-specially?)
 → boolean?
```
> Returns `#t` if `is-special-first-line?` returned `#t` for the current first line and if the buffer is scrolled down so that the first line would not (ordinarily) be visible.

```
(send a-text:first-line get-first-line-height) → number?
```
> Returns the height, in pixels, of the first line.

```
(send a-text:first-line is-special-first-line? line) → boolean?
  line : string?
```
> Override this method to control when the first line is always visible. The argument is the first line, as a string.

```
text:first-line-mixin : (class? . -> . class?)
```
  argument extends/implements: `text%`
  result implements: `text:first-line<%>`

Provides the implementation of `text:first-line<%>`. Does so by just painting the text of the first line over top of what is already there and overriding `scroll-editor-to` to patch up scrolling and `on-event` to patch up mouse handling.

```
(send a-text:first-line on-paint before?
                                 dc
                                 left
                                 top
                                 right
                                 bottom
                                 dx
                                 dy
                                 draw-caret) → void?
  before? : any/c
  dc : (is-a?/c dc<%>)
  left : real?
  top : real?
  right : real?
  bottom : real?
  dx : real?
  dy : real?
  draw-caret : (one-of/c 'no-caret 'show-inactive-caret 'show-caret)
```

Overrides on-paint in editor<%>.

Based on the various return values of the methods in text:first-line, draws the first actual line of the editor over top of the first visible line in the editor.

```
(send a-text:first-line on-event event) → void?
  event : (is-a?/c mouse-event%)
```

Overrides on-event in editor<%>.

Clicks in the first line cause the editor to scroll to the actual first line.

```
(send a-text:first-line scroll-editor-to localx
                                         localy
                                         width
                                         height
                                         refresh?
                                         bias)    → void?
  localx : real?
  localy : real?
  width : (and/c real? (not/c negative?))
  height : (and/c real? (not/c negative?))
  refresh? : any/c
  bias : (one-of/c 'start 'end 'none)
```

Overrides scroll-editor-to in editor<%>.

Scrolls a little bit more, when a scroll would be requested that scrolls something so that it is line underneath the first line.

`text:foreground-color<%>` : `interface?`
  implements: `text:basic<%>`
               `editor:standard-style-list<%>`

`text:foreground-color-mixin` : `(class? . -> . class?)`
  argument extends/implements: `text:basic<%>`
                                `editor:standard-style-list<%>`
  result implements: `text:foreground-color<%>`

This mixin changes the default text style to have the foreground color controlled by `editor:set-default-font-color`.

(send `a-text:foreground-color` `default-style-name`) $\rightarrow$ `string`

Overrides `default-style-name` in `editor<%>`.

Returns the result of `editor:get-default-color-style-name`.

(send `a-text:foreground-color` `get-fixed-style`)
 $\rightarrow$ `(is-a?/c style<%>)`

Overrides `get-fixed-style` in `text:basic<%>`.

Returns the style named by `editor:get-default-color-style-name`.

`text:hide-caret/selection<%>` : `interface?`
  implements: `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.

`text:hide-caret/selection-mixin` : `(class? . -> . class?)`
  argument extends/implements: `text:basic<%>`
  result implements: `text:hide-caret/selection<%>`

(send *a-text:hide-caret/selection* `after-set-position`) → `void`

> Augments `after-set-position` in `text%`.

> Calls `hide-caret` to hide the caret when there is only a caret and no selection.

`text:nbsp->space<%>` : `interface?`
  implements: `text%`

Classes that implement this interface silently change non-breaking spaces, ie the character (`integer->char` `160`), to regular spaces when inserted into the editor.

`text:nbsp->space-mixin` : (`class?` . `->` . `class?`)
  argument extends/implements: `text%`
  result implements: `text:nbsp->space<%>`

(send *a-text:nbsp->space* `on-insert` *start*
                                       end) → `void`
  start : `exact-int`
  end : `exact-int`

> Augments `on-insert` in `text%`.

> Starts an edit-sequence by calling `begin-edit-sequence`.

(send *a-text:nbsp->space* `after-insert` *start*
                                            len) → `void`
  start : `exact-nonnegative-integer?`
  len : `exact-nonnegative-integer?`

> Augments `after-insert` in `text%`.

> Replaces all non-breaking space characters (`integer->char` `160`) by `#\space` characters.

> Ends the edit sequence (by calling `end-edit-sequence`) started in `on-insert`.

`text:normalize-paste<%>` : `interface?`
  implements: `text:basic<%>`

```
(send a-text:normalize-paste ask-normalize?) → boolean?
```

Prompts the user if the pasted text should be normalized (and updates various preferences based on the response).

Override this method in the mixin to avoid all GUI and preferences interactions.

```
(send a-text:normalize-paste string-normalize s) → string?
  s : string?
```

Normalizes *s*. Defaults to `string-normalize-nfkc`.

```
text:normalize-paste-mixin : (class? . -> . class?)
  argument extends/implements: text:basic<%>
  result implements: text:normalize-paste<%>
```

```
(send a-text:normalize-paste do-paste start
                                       time) → void?
  start : exact-nonnegative-integer?
  time : (and/c exact? integer?)
```

Overrides `do-paste` in `text%`.

Overridden to detect when insertions are due to pasting. Sets some internal state and calls the super.

```
(send a-text:normalize-paste on-insert start
                                       len)  → void?
  start : exact-nonnegative-integer?
  len : exact-nonnegative-integer?
```

Augments `on-insert` in `text%`.

Calls `begin-edit-sequence`.

```
(send a-text:normalize-paste after-insert start
                                          len)  → void?
  start : exact-nonnegative-integer?
  len : exact-nonnegative-integer?
```

Augments `after-insert` in `text%`.

Normalizes any next text and calls `end-edit-sequence`.

```
text:searching<%> : interface?
```

implements: `editor:keymap<%>`
`text:basic<%>`

Any object matching this interface can be searched.

Searches using this class has a non-traditional feature for performance reasons. Specifically, multiple adjacent hits are coalesced into a single search results when bubbles are drawn. This means, for example, that searching for a space in a file with 80,000 spaces (as one file in the Racket code base has) is still tractable, since many of those spaces will be next to each other and thus there will be far fewer bubbles (the file in question has only 20,000 such bubbles).

```
(send a-text:searching set-searching-state str
                                            cs?
                                            replace-start)
 → void?
  str : (or/c false/c string?)
  cs? : boolean?
  replace-start : (or/c false/c number?)
```

If `str` is not `#f`, then this method highlights every occurrence of `str` in the editor. If `str` is `#f`, then it clears all of the highlighting in the buffer.

If `cs?` is `#f`, the search is case-insensitive, and otherwise it is case-sensitive.

If the `replace-start` argument is `#f`, then the search is not in replacement mode. If it is a number, then the first search hit after that position in the editor is where the next replacement will take place.

```
(send a-text:searching set-search-anchor position) → void?
  position : (or/c false/c number?)
```

Sets the anchor's position in the editor. Only takes effect if the `'framework:anchored-search` preference is on.

```
(send a-text:searching get-search-hit-count) → number?
```

Returns the number of hits for the search in the buffer, based on the count found last time that a search happened.

```
(send a-text:searching set-replace-start pos) → void?
  pos : (or/c false/c number?)
```

Sets the position where replacement next occurs. This is equivalent to calling `set-searching-state` with a new `replace-start` argument, but the other arguments the same as the last call to `set-searching-state`, but is more efficient (since `set-searching-state` will search the entire buffer and re-build all of the bubbles).

```
(send a-text:searching get-search-bubbles)
 → (listof (list/c (cons/c number number)
                   (list/c number number number)))
```

Returns information about the search bubbles in the editor. Each item in the outermost list corresponds to a single bubble. The pair of numbers is the range of the bubble and the triple of numbers is the color of the bubble, in RGB coordinates.

If `replace-start` has been set (via `set-replace-start`) and the closest search hit following `replace-start` does not collapse with an adjacent bubble,the result will include that bubble. If the closest search hit after `replace-start` is collpased with another bubble, then the search hit is not reflected in the result.

This method is intended for use in test suites.

```
text:searching-mixin : (class? . -> . class?)
  argument extends/implements: editor:keymap<%>
                               text:basic<%>
  result implements: text:searching<%>
```

This `text%` can be searched.

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

```
(send a-text:searching get-keymaps)
 → (list-of (instance keymap%))
```

Overrides `get-keymaps` in `editor:keymap<%>`.

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

```
(send a-text:searching after-insert start
                                     len)  → void?
  start : exact-nonnegative-integer?
  len : exact-nonnegative-integer?
```

Augments `after-insert` in `text%`.

Re-does any search now that the contents of the window have changed.

```
(send a-text:searching after-delete start
                                     len)  → void?
  start : exact-nonnegative-integer?
  len : exact-nonnegative-integer?
```

Augments `after-delete` in `text%`.

Re-does any search now that the contents of the window have changed.

`(send a-text:searching on-focus on?) → void?`
  `on? : boolean?`

Overrides `on-focus` in `editor<%>`.

Tells the frame containing the editor to search based on this editor via the `set-text-to-search` method.

`text:return<%> : interface?`
  implements: `text%`

Objects supporting this interface were created by `text:return-mixin`.

`text:return-mixin : (class? . -> . class?)`
  argument extends/implements: `text%`
  result implements: `text:return<%>`

Use this buffer to perform some special action when return is typed.

`(new text:return-mixin [return return])`
 `→ (is-a?/c text:return-mixin)`
  `return : (-> boolean?)`

`(send a-text:return on-local-char event) → void`
  `event : (is-a?/c key-event%)`

Overrides `on-local-char` in `editor<%>`.

If `key` is either return or newline, only invoke the `return` thunk (initialization argument) and do nothing else.

`text:wide-snip<%> : interface?`
  implements: `text:basic<%>`

`(send a-text:wide-snip add-wide-snip snip) → void`
  `snip : (instance snip%)`

Registers a snip in this editor to be resized when its viewing area changes. Ensures the snip is as wide as the viewing area.

This method should only be called by `add-wide-snip` in `canvas:wide-snip<%>`.

`(send a-text:wide-snip add-tall-snip snip)` → `void`
  `snip` : `(is-a?/c snip%)`

Registers a snip in this editor. It is resized when the viewing area of the editor changes.

This method should only be called by `add-tall-snip` in `canvas:wide-snip<%>`.

`text:wide-snip-mixin` : `(class? . -> . class?)`
  argument extends/implements: `text:basic<%>`
  result implements: `text:wide-snip<%>`

`text:delegate<%>` : `interface?`
  implements: `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 of 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.

`(send a-text:delegate get-delegate)`
 → `(union |#f| (instanceof text%))`

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 set-delegate delegate)` → `void`
  `delegate` : `(union |#f| (instanceof text%))`

This method sets the current delegate.

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.

`text:1-pixel-string-snip%` : `class?`
  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.

```
(send a-text:1-pixel-string-snip split position
                                       first
                                       second)  → void
```
  `position` : `exact`
  `first` : `(box (instanceof snip%))`
  `second` : `(box (instanceof snip%))`

> Overrides `split` in `snip%`.

> Fills the boxes with instance of `text:1-pixel-string-snip%`s.

```
(send a-text:1-pixel-string-snip copy)  → (instanceof snip%)
```

> Overrides `copy` in `snip%`.

> Creates and returns an instance of `text:1-pixel-string-snip%`.

```
(send a-text:1-pixel-string-snip get-extent  dc
                                             x
                                             y
                                             [w
                                              h
                                              descent
                                              space
                                              lspace
                                              rspace])  → void
```
  `dc` : `(instanceof dc<%>)`
  `x` : `real`
  `y` : `real`
  `w` : `(box (union non-negative-real-number |#f|))` = `|#f|`
  `h` : `(box (union non-negative-real-number |#f|))` = `|#f|`
  `descent` : `(box (union non-negative-real-number |#f|))` = `|#f|`
  `space` : `(box (union non-negative-real-number |#f|))` = `|#f|`
  `lspace` : `(box (union non-negative-real-number |#f|))` = `|#f|`
  `rspace` : `(box (union non-negative-real-number |#f|))` = `|#f|`

Overrides `get-extent` in `snip%`.

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.

```
(send a-text:1-pixel-string-snip insert  s
                                         len
                                         [pos]) → void
  s : string
  len : exact
  pos : exact = 0
```

Overrides `insert` in `string-snip%`.

```
(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
  y : real
  left : real
  top : real
  right : real
  bottom : real
  dx : real
  dy : real
  draw-caret : (union 'no-caret  'show-inactive-caret  'show-caret)
```

Overrides `draw` in `snip%`.

Draws black pixels for non-whitespace characters and draws nothing for whitespace characters.

`text:1-pixel-tab-snip%` : `class?`
  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.

```
(send a-text:1-pixel-tab-snip split position
                                    first
                                    second) → void
  position : exact
  first : (box (instanceof snip%))
  second : (box (instanceof snip%))
```

Overrides `split` in `snip%`.

Fills the boxes with instance of `text:1-pixel-tab-snip%`s.

```
(send a-text:1-pixel-tab-snip copy) → (instanceof snip%)
```

Overrides `copy` in `snip%`.

Creates and returns an instance of `text:1-pixel-tab-snip%`.

```
(send a-text:1-pixel-tab-snip get-extent dc
                                        x
                                        y
                                        [w
                                         h
                                         descent
                                         space
                                         lspace
                                         rspace]) → void
  dc : (instanceof dc<%>)
  x : real
  y : real
  w : (box (union non-negative-real-number |#f|)) = |#f|
  h : (box (union non-negative-real-number |#f|)) = |#f|
  descent : (box (union non-negative-real-number |#f|)) = |#f|
  space : (box (union non-negative-real-number |#f|)) = |#f|
  lspace : (box (union non-negative-real-number |#f|)) = |#f|
  rspace : (box (union non-negative-real-number |#f|)) = |#f|
```

Overrides `get-extent` in `snip%`.

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.

178

```
(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
  y : real
  left : real
  top : real
  right : real
  bottom : real
  dx : real
  dy : real
  draw-caret : (union 'no-caret 'show-inactive-caret 'show-caret)

> Overrides draw in snip%.

> Draws nothing.

---

text:delegate-mixin : (class? . -> . class?)
  argument extends/implements: text:basic<%>
  result implements: text:delegate<%>

This mixin provides an implementation of the text:delegate<%> interface.

```
(send a-text:delegate highlight-range  start
                                       end
                                       color
                                      [caret-space
                                       priority
                                       style])      → (-> void)
```
  start : exact-integer
  end : exact-nonnegative-integer?
  color : (or/c string? (is-a?/c color%))
  caret-space : boolean? = #f
  priority : (symbols 'high 'low) = 'low
  style : (symbols 'rectangle 'ellipse 'hollow-ellipse 'dot)
          = 'rectangle

Overrides `highlight-range` in `text:basic<%>`.

In addition to calling the super method, `highlight-range`, this method forwards the highlighting to the delegatee.

```
(send a-text:delegate unhighlight-range  start
                                         end
                                         color
                                        [caret-space
                                         style])        → void?
  start : exact-nonnegative-integer?
  end : exact-nonnegative-integer?
  color : (or/c string? (is-a?/c color%))
  caret-space : boolean? = #f
  style : (symbols 'rectangle 'ellipse 'hollow-ellipse)
        = 'rectangle
```

Overrides `unhighlight-range` in `text:basic<%>`.

This method propagates the call to the delegate and calls the super method.

```
(send a-text:delegate on-paint before?
                               dc
                               left
                               top
                               right
                               bottom
                               dx
                               dy
                               draw-caret) → void
  before? : any/c
  dc : (is-a?/c dc<%>)
  left : real?
  top : real?
  right : real?
  bottom : real?
  dx : real?
  dy : real?
  draw-caret : (one-of/c 'no-caret  'show-inactive-caret  'show-caret)
```

Overrides `on-paint` in `editor<%>`.

Draws a blue region in the delegatee editor that shows where the visible region of the delegate editor is.

```
(send a-text:delegate on-edit-sequence) → void
```

Augments `on-edit-sequence` in `editor<%>`.

starts an edit sequence in the delegate.

```
(send a-text:delegate after-edit-sequence) → void
```

Augments `after-edit-sequence` in `editor<%>`.

ends an edit sequence in the delegate.

```
(send a-text:delegate resized snip
                              redraw-now?) → void
  snip : (is-a?/c snip%)
  redraw-now? : boolean?
```

Overrides `resized` in `editor<%>`.

Sends a message to the delegate to update the size of the copied snip, if there is one.

```
(send a-text:delegate after-insert start
                                   len) → void
  start : number
  len : number
```

Augments `after-insert` in `text%`.

forwards the change to the delegate

```
(send a-text:delegate after-delete start
                                   len) → void
  start : number
  len : number
```

Augments `after-delete` in `text%`.

forwards the change to the delegate.

```
(send a-text:delegate after-change-style start
                                         len) → void
  start : number
  len : number
```

Augments `after-change-style` in `text%`.

forwards the changed style to the delegate.

```
(send a-text:delegate on-load-file filename
                                   format) → void
  filename : string
  format : symbol?
```

Augments `on-load-file` in `editor<%>`.

remembers the filename, for use in `after-load-file`.

`(send` *a-text:delegate* `after-load-file` *success?*`)` → `void`
  success? : `boolean?`

> Augments `after-load-file` in `editor<%>`.

> updates the delegate with the new contents of the text.

`text:info<%>` : `interface?`
  implements: `text:basic<%>`

Objects supporting this interface are expected to send information about themselves to the frame that is displaying them.

`text:info-mixin` : `(class? . -> . class?)`
  argument extends/implements: `editor:keymap<%>`
                               `text:basic<%>`
  result 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`.

`(send` *a-text:info* `set-anchor` *on?*`)` → `void`
  on? : `any/c`

> Overrides `set-anchor` in `text%`.

> 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.

`(send` *a-text:info* `set-overwrite-mode` *on?*`)` → `void`
  on? : `any/c`

> Overrides `set-overwrite-mode` in `text%`.

> 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 canvas's `top-level-window<%>` as the frame.

`(send` *a-text:info* `after-set-position)` → `void`

> Augments `after-set-position` in `text%`.

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.

(send *a-text:info* `after-insert` *start* *len*) → `void`
  start : `exact-nonnegative-integer?`
  len : `exact-nonnegative-integer?`

Augments `after-insert` in `text%`.

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.

(send *a-text:info* `after-delete` *start* *len*) → `void`
  start : `exact-nonnegative-integer?`
  len : `exact-nonnegative-integer?`

Augments `after-delete` in `text%`.

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.

`text:clever-file-format<%>` : `interface?`
  implements: `text%`

Objects supporting this interface are expected to support a clever file format when saving.

`text:clever-file-format-mixin` : (`class?` . -> . `class?`)
  argument extends/implements: `text%`
  result 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.

(send *a-text:clever-file-format* `on-save-file` *filename*
                                                format) → `void`
  filename : `path?`

183

format : (one-of/c 'guess 'standard 'text 'text-force-cr 'same 'copy)

Augments on-save-file in editor<%>.

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 changes to the file format only happen if the argument file-format is 'copy or 'same.

---

text:file<%> : interface?
  implements: editor:file<%>
           text:basic<%>

Mixins that implement this interface lock themselves when the file they are editing is read only.

(send *a-text:file* get-read-write?) → boolean?

Indicates whether or not this editor is in read-write mode.

(send *a-text:file* while-unlocked *thunk*) → any/c
  thunk : (-> any/c)

Unlocks the editor, calls the thunk, and then relocks the editor, all using a dynamic-wind.

---

text:file-mixin : (class? . -> . class?)
  argument extends/implements: editor:file<%>
                     text:basic<%>
  result implements: text:file<%>

(send *a-text:file* can-insert? *start* *len*) → boolean?
  start : number
  len : number

Augments can-insert? in text%.

Returns false if the result of get-read-write? is true, otherwise returns the result of calling inner.

(send *a-text:file* can-delete? *start* *len*) → boolean?

184

start : number
len : number

> Augments `can-delete?` in `text%`.
>
> Returns false if the result of `get-read-write?` is true, otherwise returns the result of calling `inner`.

`(send` *a-text:file* `after-save-file)` → `void`

> Augments `after-save-file` in `editor<%>`.
>
> 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).

`(send` *a-text:file* `after-load-file)` → `void`

> Augments `after-load-file` in `editor<%>`.
>
> 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).

`text:ports<%>` : `interface?`

Classes implementing this interface (via the associated mixin) support input and output ports that read from the editor.

There are two input ports: the normal input port just reads from the editor's contents directly and the box input port inserts an editor snip into this text and uses input typed into the box as input into the port.

They create three threads to mediate access to the input and output ports (one for each input port and one for all of the output ports).

`(send` *a-text:ports* `delete/io` *start* *end*`)` → `void`
  start : `exact-integer`
  end : `exact-integer`

> Deletes the text between *start* and *end* without changing the behavior of the ports (otherwise, deleting the text would break internal invariants of the port).

Both *start* and *end* must be less than `get-insertion-point` (or else it is safe to delete them so you don't need this method).

(send *a-text:ports* `get-insertion-point`) → `exact-integer`

Returns the position where characters put into the output port will appear.

(send *a-text:ports* `set-insertion-point` *ip*) → `void`
  ip : `exact-integer`

Sets the position where the output port will insert characters. See also `get-insertion-point`.

(send *a-text:ports* `get-unread-start-point`) → `exact-integer`

Returns the position where input will be taken into the input port (after the next time return is typed).

(send *a-text:ports* `set-unread-start-point` *usp*) → `void`
  usp : `exact-integer`

Sets the position where input will be taken into the input port (after the next time return is typed).

See also `get-unread-start-point`.

(send *a-text:ports* `set-allow-edits` *allow-edits?*) → `void`
  allow-edits? : `boolean?`

Enables or disables editing in the buffer. Be sure to update the unread start point (via `set-unread-start-point`) and the insertion point (via `set-insertion-point`) after making changes to the buffer.

(send *a-text:ports* `get-allow-edits`) → `boolean?`

Indicates if editing is allowed in the buffer at this point.

(send *a-text:ports* `insert-between` *str*) → `void`
  str : (union `snip%` `string`)

Inserts some text between the unread start point and the insertion point (and updates them properly). To insert before the two points, see `insert-before`.

See also `set-unread-start-point` and `set-insertion-point`.

(send *a-text:ports* `insert-before` *str*) → `void`
  str : (union `snip%` `string`)

Inserts some text before the unread start point and updates it and the insertion point properly. To insert between the two points, see `insert-between`.

See also `set-unread-start-point` and `set-insertion-point`.

```
(send a-text:ports submit-to-port? key) → boolean?
  key : char
```
>
> Augment this method to help control when characters should be submitted to the input port.
>
> Return `#t` or the result of calling `inner`.

```
(send a-text:ports on-submit) → void
```
>
> This method is called when text is sent into the input port.
>
> Does nothing.

```
(send a-text:ports send-eof-to-in-port) → void
```
>
> This method puts an eof into the input port.

```
(send a-text:ports send-eof-to-box-in-port) → void
```
>
> This method puts an eof into the box input port.

```
(send a-text:ports reset-input-box) → void
```
>
> This method removes the current input box from the editor (and all input in it is lost).

```
(send a-text:ports clear-output-ports) → void
```
>
> Flushes all of the data in all of the output ports that hasn't appeared in the editor yet.

```
(send a-text:ports clear-input-port) → void
```
>
> Flushes all of the data in the input port that hasn't yet been read. Reading will now block.

```
(send a-text:ports clear-box-input-port) → void
```
>
> Flushes all of the data in the box input port that hasn't yet been read. Reading will now block.

```
(send a-text:ports get-out-style-delta)
 → (or/c (is-a?/c style-delta%) string?)
```
>
> The result of this method is the style that is used to color text submitted to the result of `get-out-port`.
>
> If the result is a string that is not mapped in the editor's style list, the style named `"Standard"` is used and if that isn't mapped, the style named `"Basic"` is used.
>
> This method is called during the initialization of the class.

Defaultly returns `"text:ports out"` which is mapped to a blue style in the style list returned by `editor:get-standard-style-list`.

`(send a-text:ports get-err-style-delta)`
→ `(or/c (is-a?/c style-delta%) string?)`

The result of this method is the style that is used to color text submitted to the result of `get-err-port`.

If the result is a string that is not mapped in the editor's style list, the style named `"Standard"` is used and if that isn't mapped, the style named `"Basic"` is used.

This method is called during the initialization of the class.

Defaultly returns `"text:ports err"` which is mapped to a red italic style in the style list returned by `editor:get-standard-style-list`.

`(send a-text:ports get-value-style-delta)`
→ `(or/c (is-a?/c style-delta%) string?)`

The result of this method is the style (or the name of the style) that is used to color text submitted to the result of `get-value-port`.

If the result is a string that is not mapped in the editor's style list, the style named `"Standard"` is used and if that isn't mapped, the style named `"Basic"` is used.

This method is called during the initialization of the class.

Defaultly returns `"text:ports value"` which is mapped to a blue style in the style list returned by `editor:get-standard-style-list`.

`(send a-text:ports get-in-port)` → `input-port`

Returns the input port that data in this editor is sent to.

`(send a-text:ports get-in-box-port)` → `input-port`

Returns the box input port that data in this editor is sent to.

`(send a-text:ports get-out-port)` → `output-port`

Returns an output port that writes into this editor. The only difference between this port and the ports returned by `get-err-port` and `get-value-port` is the font style and color.

`(send a-text:ports get-err-port)` → `output-port`

Returns an output port that writes into this editor. The only difference between this port and the ports returned by `get-err-port` and `get-out-port` is the font style and color.

`(send a-text:ports get-value-port)` → `output-port`

Returns an output port that writes into this editor. The only difference between this port and the ports returned by `get-err-port` and `get-out-port` is the font style and color.

`(send` *a-text:ports* `after-io-insertion)` → `void`

This method is called after an insertion due to IO occurs.

`(send` *a-text:ports* `get-box-input-editor-snip%)`
→ `(subclass editor-snip%)`

The result of this method is used as the class of editor snips that is inserted by the box port in this editor.

`(send` *a-text:ports* `get-box-input-text%)`
→ `(is-a?/c text:input-box)`

The result of this method is instantiated and placed inside the result of `get-box-input-editor-snip%`.

`text:ports-mixin` : `(class? . -> . class?)`
   argument extends/implements: `text:wide-snip<%>`
   result implements: `text:ports<%>`

`(send` *a-text:ports* `can-insert?` *start* *len*`)` → `boolean?`
   start : `exact-integer`
   len : `exact-integer`

Augments `can-insert?` in `text%`.

Returns the results of the `inner` call, unless `get-allow-edits` returns `#f`.

`(send` *a-text:ports* `can-delete?` *start* *len*`)` → `boolean?`
   start : `exact-integer`
   len : `exact-integer`

Augments `can-delete?` in `text%`.

Returns the results of the `inner` call, unless `get-allow-edits` returns `#f`.

`(send` *a-text:ports* `on-local-char` *event*`)` → `void`
   event : `(is-a?/c key-event%)`

Overrides `on-local-char` in `editor<%>`.

Sends the data between the last position and the result of `get-unread-start-point` to the input port, unless `submit-to-port?` returns `#f`.

Also calls `on-submit`.

`(send` *a-text:ports* `on-display-size)` → `void`

> Augments `on-display-size` in `editor<%>`.

> Adjusts the embedded editor-snip (used for reading input to the `get-in-box-port`) to match the width of the editor.

`text:input-box<%>` : `interface?`
  implements: `text%`

Classes that implement this interface are used as the editors for the box input port in `text:ports%`.

`text:input-box-mixin` : `(class? . -> . class?)`
  argument extends/implements: `text%`
  result implements: `text:input-box<%>`

This mixin provides an implementation of `text:input-box<%>` for use with `text:ports<%>`.

`(send` *a-text:input-box* `on-default-char` *event*`)` → `void`
  event : `key-event%`

> Overrides `on-default-char` in `text%`.

> Notifies the `text:ports<%>` enclosing this editor that a new line of input has been provided.

`text:autocomplete<%>` : `interface?`
  implements: `text%`

The mixin implementing this interface provides an unintrusive autocompletion menu when a particular (configurable) keystroke is pressed.

`(send` *a-text:autocomplete* `auto-complete)` → `void`

> Starts a completion.

`(send` *a-text:autocomplete* `get-autocomplete-border-color)`
 → `(or/c string? (is-a?/c color%))`

The border color for the autocomplete menu. Defaults to `"black"`.

```
(send a-text:autocomplete get-autocomplete-background-color)
 → (or/c string? (is-a?/c color%))
```

The background color for the non-selected menu items. Defaults to `"laven-der"`.

```
(send a-text:autocomplete get-autocomplete-selected-color)
 → (or/c string? (is-a?/c color%))
```

The background color for the selected menu item. Defaults to `(make-object color% 204 153 255)`.

```
(send a-text:autocomplete completion-mode-key-event? key-event)
 → boolean?
  key-event : key-event%
```

Returns true when the key event passed to it should initiate the completions menu.

```
(send a-text:autocomplete get-all-words) → (listof string)
```

Returns the list of the words that autocompletion should choose from.

```
(send a-text:autocomplete get-word-at pos) → string
  pos : positive-exact-integer
```

Given an editor location, returns the prefix ending at that location that autocompletion should try to complete.

`text:autocomplete-mixin : (class? . -> . class?)`
   argument extends/implements: `text%`
   result implements: `text:autocomplete<%>`

```
(send a-text:autocomplete on-paint) → void
```

Overrides `on-paint` in `editor<%>`.

Draws the completion menu (when it is popped up).

```
(send a-text:autocomplete on-char) → void
```

Overrides `on-char` in `editor<%>`.

Takes over the handling of key events when the completions menu is visible. Also, when the completions menu is not visible, it calls the completion-mode-key-event? method to see if it should start completing.

`(send a-text:autocomplete on-event) → void`

>   Overrides `on-event` in `editor<%>`.

>   This method is overridden to allow mouse access of the completions menu. It only handles events when there is a menu open and the mouse is in the menu, in which case it makes the menu trace the mouse.

>   The only time it does not call the super method is when the mouse is button is pushed.

`text:basic% : class?`
  superclass: `(text:basic-mixin (editor:basic-mixin text%))`

`text:hide-caret/selection% : class?`
  superclass: `(text:hide-caret/selection-mixin text:basic%)`

`text:nbsp->space% : class?`
  superclass: `(text:nbsp->space-mixin text:basic%)`

`text:delegate% : class?`
  superclass: `(text:delegate-mixin text:basic%)`

`text:wide-snip% : class?`
  superclass: `(text:wide-snip-mixin text:basic%)`

`text:standard-style-list% : class?`
  superclass: `(editor:standard-style-list-mixin text:wide-snip%)`

```
text:input-box% : class?
  superclass: (text:input-box-mixin text:standard-style-list%)
```

```
text:keymap% : class?
  superclass: (editor:keymap-mixin text:standard-style-list%)
```

```
text:return% : class?
  superclass: (text:return-mixin text:keymap%)
```

```
text:autowrap% : class?
  superclass: (editor:autowrap-mixin text:keymap%)
```

```
text:file% : class?
  superclass: (text:file-mixin (editor:file-mixin text:autowrap%))
```

```
text:clever-file-format% : class?
  superclass: (text:clever-file-format-mixin text:file%)
```

```
text:backup-autosave% : class?
  superclass: (editor:backup-autosave-mixin text:clever-file-format%)
```

```
text:searching% : class?
  superclass: (text:searching-mixin text:backup-autosave%)
```

```
text:info% : class?
  superclass: (text:info-mixin (editor:info-mixin text:searching%))
```

```
(text:range? arg) → boolean?
  arg : any/c
```

Determines if `arg` is an instance of the range struct.

```
(text:range-start range) → exact-nonnegative-integer?
  range : text:range?
```

Returns the start position of the range.

```
(text:range-end range) → exact-nonnegative-integer?
  range : text:range?
```

Returns the end position of the range.

```
(text:range-caret-space? range) → boolean?
  range : text:range?
```

Returns a boolean indicating where the caret-space in the range goes. See also `highlight-range`.

```
(text:range-style range) → exact-nonnegative-integer?
  range : text:range?
```

Returns the style of the range. See also `highlight-range`.

```
(text:range-color range) → (or/c string? (is-a?/c color%))
```

```
range : text:range?
```

Returns the color of the highlighted range.

```
(text:autocomplete-append-after) → string?
(text:autocomplete-append-after suffix) → void?
  suffix : string?
```

A string that is inserted after a completion is inserted by a `text:autocomplete` instance.

Defaults to `""`.

```
(text:autocomplete-limit) → (and/c integer? exact? positive?)
(text:autocomplete-limit count) → void?
  count : (and/c integer? exact? positive?)
```

Controls the number of completions visible at a time in the menu produced by `text:autocomplete` instances.

Defaults to 15.

```
(text:get-completions/manuals manuals) → (listof string?)
  manuals : (or/c false/c (listof symbol?))
```

Returns the list of keywords for the manuals from *manuals* by extracting all of the documented exports of the manuals. The symbols are meant to be module paths, eg the quoted form of the argument to `require`.

If *manuals* is false, then all of the documented names are used.

```
(text:lookup-port-name manuals)
 → (or/c (is-a?/c editor:basic<%>) false/c)
  manuals : symbol?
```

Returns the editor instance whose port-name matches the given symbol. If no editor can be found, then returns `false`.

28 Test

```
(require framework/test)
```

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 Racket functions. For example,

```
(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. Use `(require framework/test)`.

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 GRacket.

28.1 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.

28.2 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.

28.3 Technical Issues

28.3.1 Active Frame

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

28.3.2 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 `make-eventspace` for more info.

28.3.3 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.

28.4 Test Functions

```
(test:button-push button) → void?
  button : (or/c (λ (str)
                   (and (string? str)
                        (test:top-level-focus-window-has?
                         (λ (c)
                           (and (is-a? c button%)
                                (string=? (send c get-label) str)
                                (send c is-enabled?)
                                (send c is-shown?))))))
                 (and/c (is-a?/c button%)
                        (λ (btn)
                          (and (send btn is-enabled?)
                               (send btn is-shown?)))
                        (λ (btn)
                          (test:top-level-focus-window-has?
                           (λ (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:set-radio-box! radio-box state) → void?
  radio-box : (or/c string? (is-a?/c radio-box%))
  state : (or/c 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%` object with a label match-
ing that string, otherwise it uses `radio-box` itself.

```
(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:set-check-box! check-box state) → void?
  check-box : (or/c 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!  choice  str) → void?
  choice : (or/c string? (is-a?/c choice%))
  str : (or/c string? (and/c number? exact? integer? positive?))
```

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-list-box!  choice  str) → void?
  choice : (or/c string? (is-a?/c list-box%))
  str : (or/c string? (and/c number? exact? integer? positive?))
```

Selects *list-box*'s item *str*. If *list-box* is a string, this function searches for a `list-box%` with a label matching that string, otherwise it uses *list-box* itself.

```
(test:keystroke  key  [modifier-list]) → void?
  key : (or/c char? symbol?)
  modifier-list : (listof (symbols 'alt 'control 'meta 'shift 'noalt 'nocontrol 'nometea 'nosh
                  = null
```

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 `#\return`, not `#\newline`.

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  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 button x y [modifiers]) → void?
  button : (symbols 'left 'middle 'right)
  x : (and/c exact? integer?)
  y : (and/c exact? integer?)
  modifiers : (listof (symbols 'alt 'control 'meta 'shift 'noalt 'nocontrol 'nometa 'noshift))
            = null
```

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:run-interval msec) → void?
  msec : number?
(test:run-interval) → number?
```

See also §28.1 "Actions and completeness". The first case in the case-lambda sets the run interval to *msec* milliseconds and the second returns the current setting.

```
(test:current-get-eventspaces) → (-> (listof eventspace?))
(test:current-get-eventspaces func) → void?
  func : (-> (listof eventspace?))
```

This parameter that specifies which evenspaces (see also §1.1.4 "Event Dispatching and Eventspaces") 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: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: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: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`.

```
(test:number-pending-actions) → number?
```

Returns the number of pending events (those that haven't completed yet)

```
(test:reraise-error) → void?
```

See also §28.2 "Errors".

```
(test:run-one f) → void?
  f : (-> void?)
```

Runs the function `f` as if it was a simulated event.

29 Version

```
(version:add-spec spec revision) → void?
  spec : any/c
  revision : any/c
```

These two values are appended to the version string. `write` is used to transform them to strings. For example:

```
(version:add-spec 's  1)
```

in version 205 will make the version string be `205s1`. The symbols `'f` and `'d` are used internally for framework and drscheme revisions.

```
(version:version) → string?
```

This function returns a string describing the version of this application. See also `version:add-spec`.

Index

214