Embedded GUI: Widgets Within Editors

Version 9.4.0.3

Mike T. McHenry

November 1, 2026

```
(require embedded-gui)
```
package: `gui-lib`

The `embedded-gui` library provides a class hierarchy for creating graphical boxes within `editor<%>` objects with geometry management that mirrors that of `vertical-panel%` and `horizontal-panel%`.

Contents

1 Containers **3**

1.1 aligned-pasteboard% . . . 3

1.2 alignment<%> . . . 3

1.3 alignment-parent<%> . . . 4

1.4 stretchable-snip<%> . . . 5

1.5 horizontal-alignment% . . . 6

1.6 vertical-alignment% . . . 6

1.7 dllist<%> . . . 7

2 Controls **8**

2.1 embedded-text-button% . . . 8

2.2 embedded-button% . . . 8

2.3 embedded-toggle-button% . . . 9

2.4 embedded-message% . . . 9

2.5 vline% . . . 10

2.6 hline% . . . 10

3 Control Snips **11**

3.1 snip-wrapper% . . . 11

3.2 text-button-snip% . . . 11

3.3 button-snip% . . . 11

3.4 toggle-button-snip% . . . 12

4 Helpers **14**

5 Snip Functions **17**

1 Containers

1.1 aligned-pasteboard%

```
aligned-pasteboard% : class?
  superclass: pasteboard%
  extends: alignment-parent<%>
```

Acts as the top of an alignment<%> tree.

1.2 alignment<%>

```
alignment<%> : interface?
  implements: dllist<%>
```

```
(send an-alignment get-parent) → (is-a?/c alignment-parent<%>)
```

The parent of the alignment in the tree.

```
(send an-alignment set-min-sizes) → void?
```

Tells the alignment that its sizes should be calculated.

```
(send an-alignment align x-offset
                         y-offset
                         width
                         height)  → void?
  x-offset : (and/c real? (not/c negative?))
  y-offset : (and/c real? (not/c negative?))
  width : (and/c real? (not/c negative?))
  height : (and/c real? (not/c negative?))
```

Tells itself to align its children on the pasteboard in the given rectangle defined by *width*, *height* and a top left corner point given as offsets into the pasteboards top left corner.

```
(send an-alignment get-min-width)
 → (and/c real? (not/c negative?))
```

The minimum width this alignment must be.

```
(send an-alignment get-min-height)
 → (and/c real? (not/c negative?))
```

The minimum height this alignment must be.

```
(send an-alignment stretchable-width) → boolean?
(send an-alignment stretchable-width value) → void?
  value : boolean?
```

Gets/sets the property of stretchability in the x dimension.

```
(send an-alignment stretchable-height) → boolean?
(send an-alignment stretchable-height value) → void?
  value : boolean?
```

Gets/sets the property of stretchability in the y dimension.

```
(send an-alignment show/hide show?) → void?
  show? : boolean?
```

Tells the alignment to show or hide its children.

```
(send an-alignment show show?) → void?
  show? : boolean?
```

Tells the alignment that its show state is the given value and it should show or hide its children accordingly.

1.3 alignment-parent<%>

```
alignment-parent<%> : interface?
```

```
(send an-alignment-parent get-pasteboard)
 → (is-a?/c pasteboard%)
```

The pasteboard that this alignment is being displayed to.

```
(send an-alignment-parent add-child child) → void?
  child : (is-a?/c alignment<%>)
```

Add the given alignment as a child after the existing child.

```
(send an-alignment-parent delete-child child) → void?
  child : (is-a?/c alignment<%>)
```

Deletes a child from the alignments.

```
(send an-alignment-parent is-shown?) → boolean?
```

True if the alignment is being shown (accounting for its parent being shown).

1.4 stretchable-snip<%>

```
stretchable-snip<%> : interface?
```

Must be implemented by any snip class whose objects will be stretchable when inserted into an aligned-pasteboard<%> within a snip-wrapper%.

```
(send a-stretchable-snip get-aligned-min-width)
 → (and/c real? (not/c negative?))
```

The minimum width that the snip can be resized to.

```
(send a-stretchable-snip get-aligned-min-height)
 → (and/c real? (not/c negative?))
```

The minimum height that the snip can be resized to.

```
(send a-stretchable-snip stretchable-width) → boolean?
(send a-stretchable-snip stretchable-width stretch?) → void?
  stretch? : boolean?
```

Gets/sets whether or not the snip can be stretched in the X dimension.

```
(send a-stretchable-snip stretchable-height) → boolean?
(send a-stretchable-snip stretchable-height stretch?) → void?
  stretch? : boolean?
```

Gets/sets whether or not the snip can be stretched in the Y dimension.

1.5 horizontal-alignment%

```
horizontal-alignment% : class?
  superclass: dllist<%>
  extends: alignment<%>
           alignment-parent<%>
```

```
 (new horizontal-alignment%
    [parent parent]
   [[show? show?]
    [after after]])
 → (is-a?/c horizontal-alignment%)
  parent : (is-a?/c alignment-parent<%>)
  show? : boolean? = #t
  after : (or/c (is-a?/c alignment<%>) false/c) = #f
```

Inserts a new horizontal-alignment container into *parent*—optionally after a given container also in *parent*. The new container can be initially shown or hidden.

1.6 vertical-alignment%

```
vertical-alignment% : class?
  superclass: dllist<%>
  extends: alignment<%>
           alignment-parent<%>
```

```
 (new vertical-alignment%
    [parent parent]
   [[show? show?]
    [after after]])
 → (is-a?/c vertical-alignment%)
```

```
  parent : (is-a?/c alignment-parent<%>)
  show? : boolean? = #t
  after : (or/c (is-a?/c alignment<%>) false/c) = #f
```

Inserts a new vertical-alignment container into *parent*—optionally after a given container also in *parent*. The new container can be initially shown or hidden.

1.7 dllist<%>

```
dllist<%> : interface?
```

Defines a doubly-linked.

```
(send a-dllist next) → (is-a?/c dllist<%>)
(send a-dllist next new-next) → void?
  new-next : (is-a?/c dllist<%>)
```

Gets/sets the next field to be the given dllist.

```
(send a-dllist prev) → (is-a?/c dllist<%>)
(send a-dllist prev new-prev) → void?
  new-prev : (is-a?/c dllist<%>)
```

Gets/sets the previous item in the list.

```
(send a-dllist for-each f) → void?
  f : ((is-a?/c dllist<%>) . -> . void?)
```

Applies *f* to every element of the dllist.

```
(send a-dllist map-to-list f) → (listof any/c)
  f : ((is-a?/c dllist<%>) . -> . any/c)
```

Creates a Racket list by applying *f* to every element of *a-dllist*.

2 Controls

2.1 embedded-text-button%

```
embedded-text-button% : class?
  superclass: snip-wrapper%
  extends: alignment<%>
```

A button with a text label.

```
(new embedded-text-button%
   [label label]
   [callback callback])
→ (is-a?/c embedded-text-button%)
 label : string?
 callback : ((is-a?/c text-button-snip%) (is-a?/c event%) . -> . void?)
```

The *callback* is called when the button is clicked.

2.2 embedded-button%

```
embedded-button% : class?
  superclass: snip-wrapper%
  extends: alignment<%>
```

A clickable button with a bitmap label.

```
(new embedded-button%
   [images images]
   [callback callback])
→ (is-a?/c embedded-button%)
 images : (cons/c path-string? path-string?)
 callback : ((is-a?/c button-snip%) (is-a?/c event%) . -> . void?)
```

The *images* argument is a pair filenames to be load as the button-label image, where the first is the image for when the button is at rest, and the second is the image for the button while its pressed.

The *callback* is called when the button is clicked.

2.3 embedded-toggle-button%

```
embedded-toggle-button% : class?
  superclass: snip-wrapper%
  extends: alignment<%>
```

A check-box%-like control that a user can toggle between checked and unchecked states.

```
(new embedded-toggle-button%
     [images-off images-off]
     [images-on images-on]
     [turn-on turn-on]
     [turn-off turn-off]
    [[state state]])
→ (is-a?/c embedded-toggle-button%)
 images-off : (cons/c path-string? path-string?)
 images-on : (cons/c path-string? path-string?)
 turn-on : ((is-a?/c toggle-button-snip%) (is-a?/c event%) . -> . void?)
 turn-off : ((is-a?/c toggle-button-snip%) (is-a?/c event%) . -> . void?)
 state : (symbols 'on 'off) = 'on
```

The *images-off* argument is a pair filenames to be load as the button-label image, where the first is the image for when the button is at rest, and the second is the image for the button while its pressed—in both cases when the button is not checked by the user. The *images-on* argument similarly determines the images for then the button is checked.

The *turn-on* and *turn-off* callbacks are invoked when the button changes to checked or unchecked, respectively.

The *state* argument determines whether the button is initially checked.

2.4 embedded-message%

```
embedded-message% : class?
  superclass: snip-wrapper%
```

A static text label.

```
(new embedded-message%
    [parent parent]
    [label label])
→ (is-a?/c embedded-message%)
 parent : (is-a?/c alignment-parent<%>)
 label : string?
```

Creates a static control that displays *label*.

2.5 vline%

```
vline% : class?
  superclass: snip-wrapper%
  extends: alignment<%>
```

Displays a vertical line across the region that is inserted into.

```
(new vline% [parent parent]) → (is-a?/c vline%)
  parent : (is-a?/c alignment-parent<%>)
```

2.6 hline%

```
hline% : class?
  superclass: snip-wrapper%
  extends: alignment<%>
```

Displays a horizontal line across the region that is inserted into.

```
(new hline% [parent parent]) → (is-a?/c hline%)
  parent : (is-a?/c alignment-parent<%>)
```

3 Control Snips

To allow the buttons to be pushed, the editor in which they appear must forward clicks to them properly.

3.1 snip-wrapper%

```
snip-wrapper% : class?
  superclass: dllist<%>
  extends: alignment<%>
```

Adapts an arbitrary snip<%> to work in an alignment container.

```
(new snip-wrapper%
   [parent parent]
   [snip snip])     → (is-a?/c snip-wrapper%)
 parent : (is-a?/c alignment-parent<%>)
 snip : (is-a?/c snip%)
```

Adds *snip* to *parent*.

3.2 text-button-snip%

```
text-button-snip% : class?
  superclass: snip%
```

A button with a text label.

```
(new text-button-snip%
   [label label]
   [callback callback])
→ (is-a?/c text-button-snip%)
 label : string?
 callback : ((is-a?/c text-button-snip%) (is-a?/c event%) . -> . void)
```

The *callback* is called when the button is clicked.

3.3 button-snip%

```
button-snip% : class?
```

superclass: snip%

A clickable button with a bitmap label.

```
(new button-snip%
   [images images]
   [callback callback])
→ (is-a?/c button-snip%)
 images : (cons/c path-string? path-string?)
 callback : ((is-a?/c button-snip%) (is-a?/c event%) . -> . void?)
```

The *images* argument is a pair filenames to be load as the button-label image, where the first is the image for when the button is at rest, and the second is the image for the button while its pressed.

The *callback* is called when the button is clicked.

3.4 toggle-button-snip%

```
toggle-button-snip% : class?
  superclass: snip%
```

A check-box%-like control that a user can toggle between checked and unchecked states.

```
(new toggle-button-snip%
    [images-off images-off]
    [images-on images-on]
    [turn-on turn-on]
    [turn-off turn-off]
   [[state state]])
→ (is-a?/c toggle-button-snip%)
 images-off : (cons/c path-string? path-string?)
 images-on : (cons/c path-string? path-string?)
 turn-on : ((is-a?/c toggle-button-snip%) (is-a?/c event%) . -> . void?)
 turn-off : ((is-a?/c toggle-button-snip%) (is-a?/c event%) . -> . void?)
 state : (symbols 'on 'off) = 'on
```

The *images-off* argument is a pair filenames to be load as the button-label image, where the first is the image for when the button is at rest, and the second is the image for the button while its pressed—in both cases when the button is not checked by the user. The *images-on* argument similarly determines the images for then the button is checked.

The *turn-on* and *turn-off* callbacks are invoked when the button changes to checked or unchecked, respectively.

The *state* argument determines whether the button is initially checked.

4 Helpers

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

Extends an editor snip the `stretchable-snip<%>` interface, which allows it to be stretched to fit an `alignment-parent<%>`'s allotted width. Stretchable snips are useful as the snip of a `snip-wrapper%`

```
stretchable-editor-snip% : class?
  superclass: editor-snip%
  extends: stretchable-editor-snip-mixin
           editor-snip%
```

```
(new stretchable-editor-snip%
   [[stretchable-width stretchable-width]
    [stretchable-height stretchable-height]])
 → (is-a?/c stretchable-editor-snip%)
  stretchable-width : boolean? = #t
  stretchable-height : boolean? = #t
```

Creates a stretchable snip with the given initial stretchability.

```
(fixed-width-label-snip possible-labels) → (subclass?/c snip%)
  possible-labels : (listof string?)
```

Returns a subclass of `snip%` that takes a single initialization argument. The argument provided when instantiating the class must be a member of *possible-labels*; the given label is displayed by the snip, but the snip is sized to match the longest of the labels in *possible-labels*.

In other words, the resulting class helps align multiple GUI elements t hat are labeled from a particular set of strings.

```
tabbable-text<%> : interface?
```

An interface for tabbing between embedded `text%`s.

(send *a-tabbable-text* set-caret-owner) → void?

Moves the caret into the tabbable-text<%>.

(send *a-tabbable-text* set-ahead) → void?

Called when tabbing ahead.

(send *a-tabbable-text* set-back) → void?

Called when tabbing backward.

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

Adds the tabbable-text<%> interface to an editor:text% class, where instantiation installs key bindings to tab ahead and backward

(set-tabbing *a-text* ...) → void?
a-text : (is-a?/c tabbable-text<%>)

Sets the tabbing order of tabbable-text<%>s by setting each text's set-ahead and set-back thunks to point to its neighbor in the argument list.

grey-editor-snip-mixin : (class? . -> . class?)
argument extends/implements: editor-snip%

Gives an editor-snip% a colored background indicating that is disabled. The editor is not disabled by the mixin however, and must be locked separately.

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

Gives an editor<%> a colored background indicating that is disabled. The editor is not disabled by the mixin however, and must be locked separately.

```
single-line-text-mixin : (class? . -> . class?)
```
argument extends/implements: `editor:keymap<%>`

Restricts a text to one line by overriding its key bindings to do nothing on enter.

```
cue-text-mixin : (class? . -> . class?)
```
argument extends/implements: `text%`

Gives a `text%` an instantiation argument of a string that is displayed in the `text%` initially in grey; the text disappears when the text gets focus. This technique is useful for labeling texts without needing to take up space.

```
cue-text% : class?
```
superclass: `(cue-text-mixin text%)`

```
(new cue-text%
     [[cue-text cue-text]
      [color color]
      [behavior behavior]]) → (is-a?/c cue-text%)
  cue-text : string? = ""
  color : string? = "gray"
  behavior : (listof (one-of/c 'on-focus 'on-char))
           = '(on-focus)
```

Creates an instance with the given initial content, color, and behvior for when to clear the text.

```
(send a-cue-text clear-cue-text) → void?
```

Clears the cue text, if it's still present.

5 Snip Functions

```
(snip-width snip) → real?
  snip : (is-a?/c snip%)
```

The width of a snip in the parent pasteboard.

```
(snip-height snip) → real?
  snip : (is-a?/c snip%)
```

The height of a snip in the parent pasteboard.

```
(snip-min-width snip) → real?
  snip : (is-a?/c snip%)
```

The minimum width of the snip

```
(snip-min-height snip) → real?
  snip : (is-a?/c snip%)
```

The minimum height of the snip.

```
(snip-parent snip) → (is-a?/c pasteboard%)
  snip : (is-a?/c snip%)
```

The pasteboard that contains the snip.

```
(fold-snip f init-acc snip) → any/c
  f : ((is-a?/c snip%) any/c . -> . any/c)
  init-acc : any/c
  snip : (is-a?/c snip%)
```

Applies *f* to all snips in the parent of *snip*, starting with *snip*.

```
(for-each-snip f first-snip more ...) → void?
  f : ((is-a?/c snip%) . -> . any/c)
  first-snip : (is-a?/c snip%)
  more : list?
```

Applies the function to each snip in the parent of *first-snip*, starting with *first-snip*. If *more* lists are supplied, they are used for extra arguments to *f*, just like extra lists provided to for-each.

```
(map-snip f first-snip more ...) → void?
  f : ((is-a?/c snip%) . -> . any/c)
  first-snip : (is-a?/c snip%)
  more : list?
```

Applies the function to each snip in the parent of *first-snip*, starting with *first-snip*, and accumulates the results into a list. If *more* lists are supplied, they are used for extra arguments to *f*, just like extra lists provided to map.

```
(stretchable-width? snip) → boolean?
  snip : (is-a?/c snip%)
```

True if the snip can be resized in the X dimension.

```
(stretchable-height? snip) → boolean?
  snip : (is-a?/c snip%)
```

True if the snip can be resized in the Y dimension.