MrLib: Extra GUI Libraries

Version 6.1.1

November 4, 2014

Contents

Index **53**

1 Aligned Pasteboard

The aligned-pasteboard library provides classes derived from pasteboard% with geometry management that mirrors that of vertical-panel% and horizontal-panel%.

```
(require mrlib/aligned-pasteboard)
```
package: gui-lib

1.1 aligned-pasteboard<%>

```
aligned-pasteboard<%> : interface?
```

```
(send an-aligned-pasteboard get-aligned-min-height) → real?
```

The minimum height an aligned-pasteboard can be and still fit the heights of all of its children.

```
(send an-aligned-pasteboard get-aligned-min-width) → real?
```

The minimum width an aligned-pasteboard can be and still fit the widths of all of its children.

```
(send an-aligned-pasteboard realign width
                                    height) → void?
  width : exact-nonnegative-integer?
  height : exact-nonnegative-integer?
(send an-aligned-pasteboard realign) → void?
```

Realigns the children inside the aligned-pasteboard<%> to either a given *width* and *height* or the previously alloted width and height.

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

Calculates the minimum width and height of the of the pasteboard based on children's min-sizes and stores it for later retrieval via the getters.

1.2 horizontal-pasteboard%

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

```
(new horizontal-pasteboard% ...superclass-args...)
 → (is-a?/c horizontal-pasteboard%)
```

Passes all arguments to `super-init`.

```
(send a-horizontal-pasteboard after-delete snip) → void?
  snip : (is-a?/c snip%)
```

Overrides `after-delete` in `pasteboard%`.

```
(send a-horizontal-pasteboard after-insert snip
                                           before
                                           x
                                           y)     → void?
  snip : (is-a?/c snip%)
  before : (or/c (is-a?/c snip%) false/c)
  x : real?
  y : real?
```

Overrides `after-insert` in `pasteboard%`.

```
(send a-horizontal-pasteboard after-reorder snip
                                            to-snip
                                            before?) → boolean?
  snip : (is-a?/c snip%)
  to-snip : (is-a?/c snip%)
  before? : any/c
```

Overrides `after-reorder` in `pasteboard%`.

```
(send a-horizontal-pasteboard resized snip
                                      redraw-now?) → void?
  snip : (is-a?/c snip%)
  redraw-now? : any/c
```

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

1.3 vertical-pasteboard%

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

```
(new vertical-pasteboard% ...superclass-args...)
 → (is-a?/c vertical-pasteboard%)
```

Passes all arguments to `super-init`.

```
(send a-vertical-pasteboard after-delete snip) → void?
  snip : (is-a?/c snip%)
```

Overrides `after-delete` in `pasteboard%`.

```
(send a-vertical-pasteboard after-insert snip
                                         before
                                         x
                                         y)     → void?
  snip : (is-a?/c snip%)
  before : (or/c (is-a?/c snip%) false/c)
  x : real?
  y : real?
```

Overrides `after-insert` in `pasteboard%`.

```
(send a-vertical-pasteboard after-reorder snip
                                          to-snip
                                          before?) → boolean?
  snip : (is-a?/c snip%)
  to-snip : (is-a?/c snip%)
  before? : any/c
```

Overrides `after-reorder` in `pasteboard%`.

```
(send a-vertical-pasteboard resized snip
                                    redraw-now?) → void?
  snip : (is-a?/c snip%)
  redraw-now? : any/c
```

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

1.4 aligned-editor-snip%

```
aligned-editor-snip% : class?
  superclass: editor-snip%
```

Calls the `realign` method when resized.

1.5 aligned-editor-canvas%

```
aligned-editor-canvas% : class?
  superclass: editor-canvas%
```

Calls the realign method when resized.

1.6 aligned-pasteboard-parent<%>

```
aligned-pasteboard-parent<%> : interface?
```

This interface must be implemented by any class who's editor is an aligned-pasteboard<%>.

```
(send an-aligned-pasteboard-parent set-aligned-min-sizes)
 → void?
```

1.7 stretchable-snip<%>

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

This interface must be implemented by any snip class who's objects will be stretchable when inserted into an aligned-pasteboard<%>.

```
(send a-stretchable-snip get-aligned-min-height) → real?
```

The minimum height that the snip can be resized to

```
(send a-stretchable-snip get-aligned-min-width) → real?
```

The minimum width that the snip can be resized to.

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

Whether or not the snip can be stretched in the Y dimension

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

Whether or not the snip can be stretched in the X dimension

2 Bitmap Label

```
(require mrlib/bitmap-label)
```
package: gui-lib

```
(make-bitmap-label str img [font]) → (is-a?/c bitmap%)
  str : string?
  img : (or/c (is-a?/c bitmap%) path-string?)
  font : (is-a?/c font%) = normal-control-font
```

Constructs a bitmap label suitable for use a button that contains the image specified by *img* followed by the text in *str*.

```
((bitmap-label-maker str img) future-parent) → (is-a?/c bitmap%)
  str : string?
  img : (or/c (is-a?/c bitmap%) path-string?)
  future-parent : any/c
```

An older variant of make-bitmap-label that was designed to obtain a font to use from a container *future-parent*. The *future-parent* argument is currently ignored.

3 Cache-image Snip

```
(require mrlib/cache-image-snip)
```
package: gui-lib

> **NOTE:** This library is deprecated; use racket/gui, instead. This library will no longer be public in a future release; much of it will be available privately to continue to support the implementation of htdp/image, but the other exported functions here are not useful and have names that confusingly match unrelated other libraries.

The mrlib/cache-image-snip library provides the core data structure for DrRacket's "image.rkt" teachpack. Images in the "image.rkt" teachpack are instances of the cache-image-snip% class.

The library also defines a new type, argb, that represents a bitmap, but with alpha values. It has a maker, two selectors, and a predicate.

```
cache-image-snip% : class?
  superclass: image-snip%
```

The cache-image-snip% class is a subclass of image-snip% simply so that its instances can be compared with image-snip% using equal?. All image-snip% functionality is overridden or ignored.

```
(send a-cache-image-snip equal-to? snip
                                   equal?) → boolean?
  snip : (is-a?/c image-snip%)
  equal? : (any/c any/c . -> . boolean?)
```

Calls the other-equal-to? method of *snip* if it is also a cache-image-snip% instance, otherwise calls the other-equal-to? of *a-cache-image-snip*.

```
(send a-cache-image-snip get-argb) → argb?
```

Returns a pixel array for this image, forcing it to be computed.

```
(send a-cache-image-snip get-argb-proc)
 → (argb? exact-integer? exact-integer? . -> . void?)
```

Returns a procedure that fills in an argb with the contents of this image at the given offset

```
(send a-cache-image-snip get-argb/no-compute)
 → (or/c false/c argb?)
```

Returns a pixel array for this image or `#f` if it has not been computed yet.

```
(send a-cache-image-snip get-bitmap)
 → (or/c false/c (is-a?/c bitmap%))
```

Overrides `get-bitmap` in `image-snip%`.

Builds (if not yet built) a bitmap corresponding to this snip and returns it.

If the width or the height of the snip is `0`, this method return `#f`.

```
(send a-cache-image-snip get-dc-proc)
 → (or/c false/c ((is-a?/c dc<%>) real? real? -> void?))
```

Either returns false, or a procedure that draws the contents of this snip into a dc.

```
(send a-cache-image-snip get-pinhole) → real? real?
```

Returns the pinhole coordinates for this image, counting from the top-left of the image.

```
(send a-cache-image-snip get-size)
 →
   exact-nonnegative-integer?
   exact-nonnegative-integer?
```

Returns the width and height for the image.

```
(send a-cache-image-snip other-equal-to? snip
                                         equal?) → boolean?
  snip : (is-a?/c image-snip%)
  equal? : (any/c any/c . -> . boolean?)
```

Overrides `other-equal-to?` in `image-snip%`.

Refines the comparison of `other-equal-to?` in `image-snip%` to exactly match alpha channels.

```
snip-class : (is-a?/c snip-class%)
```

This snipclass is used for saved cache image snips.

```
(make-argb vectorof width height) → argb?
  vectorof : byte?
  width : exact-nonnegative-integer?
  height : exact-nonnegative-integer?
```

Constructs a new argb value with the given width and height, using the data in the vector. The vector has four entries for each pixel, an alpha, red, green, and blue value. The pixels are specified in row-major order, so that the pixel at location (x,y) comes from vector entry (4*(x+(width*y))).

```
(argb-vector argb) → (vectorof byte?)
  argb : argb?
```

Extracts the vector from *argb*. The resulting vector has entries in row-major order, so that the data for the pixel at (x,y) winds up in four vector entries beginning at (4*(x+(width*y))).

This function is in a library that will be removed in a future version. Do not use it.

The mentions of "argb" in this library are not bytes? objects (i.e, not the same as for example, the result of get-argb-pixels)

```
(argb-width argb) → exact-nonnegative-integer?
  argb : argb?
```

Extracts the width from *argb*.

This function is in a library that will be removed in a future version. Do not use it.

The mentions of "argb" in this library are not bytes? objects (i.e, not the same as for example, the result of get-argb-pixels)

```
(argb-height argb) → exact-nonnegative-integer?
  argb : argb?
```

Extracts the height from *argb*.

This function is in a library that will be removed in a future version. Do not use it.

The mentions of "argb" in this library are not bytes? objects (i.e, not the same as for example, the result of get-argb-pixels)

```
(argb? v) → boolean?
  v : any/c
```

Returns #t if *v* is an argb, #f otherwise.

This function is in a library that will be removed in a future version. Do not use it.

The mentions of "argb" in this library are not bytes? objects (i.e, not the same as for example, the result of get-argb-pixels)

```
(overlay-bitmap dest dx dy img mask) → void?
  dest : argb?
  dx : exact-integer?
  dy : exact-integer?
  img : (is-a?/c bitmap%)
  mask : (is-a?/c bitmap%)
```

Changes argb, overlaying *img* with masking based on *mask* at (*dx*, *dy*) from the top-left.

This function is in a library that will be removed in a future version. Do not use it.

The mentions of "argb" in this library are not bytes? objects (i.e, not the same as for example, the result of get-argb-pixels)

```
(build-bitmap draw width height) → (is-a?/c bitmap%)
  draw : ((is-a?/c dc<%>) . -> . any)
  width : (integer-in 1 10000)
  height : (integer-in 1 10000)
```

Builds a bitmap of size *width* by *height*, using the procedure *draw* to render the bitmap content into the given dc<%>.

This function is in a library that will be removed in a future version. Do not use it.

The mentions of "argb" in this library are not bytes? objects (i.e, not the same as for example, the result of get-argb-pixels)

```
(flatten-bitmap bitmap) → (is-a?/c bitmap%)
  bitmap : (is-a?/c bitmap%)
```

Builds a new bitmap that flattens the original *bitmap* with its mask (as determined by get-loaded-mask in bitmap%), producing a bitmap that has no mask, and looks the way that bitmap would draw (when drawn with the mask) onto a white background.

This function is in a library that will be removed in a future version. Do not use it.

The mentions of "argb" in this library are not bytes? objects (i.e, not the same as for example, the result of get-argb-pixels)

```
(argb->cache-image-snip argb dx dy) → (is-a?/c cache-image-snip%)
  argb : argb?
  dx : real?
  dy : real?
```

Builds a new cache-image-snip% based on the contents of *argb*, using *dx* and *dy* as the pinhole.

This function is in a library that will be removed in a future version. Do not use it.

The mentions of "argb" in this library are not bytes? objects (i.e, not the same as for example, the result of get-argb-pixels)

```
(argb->bitmap argb) → (or/c false/c (is-a?/c bitmap%))
  argb : argb?
```

Builds a bitmap that draws the same way as `argb`; the alpha pixels are put into the bitmap's `get-loaded-mask` bitmap.

If the width or height of `argb` is `0`, this returns `#f`.

This function is in a library that will be removed in a future version. Do not use it.

The mentions of "argb" in this library are not `bytes?` objects (i.e, not the same as for example, the result of `get-argb-pixels`)

4 Close Icon

```
(require mrlib/close-icon)
```
package: gui-lib

The close-icon% class provides a clickable close button icon.

```
close-icon% : class?
  superclass: canvas%
```

```
(new close-icon%
     [parent parent]
    [[callback callback]
     [bg-color bg-color]
     [horizontal-pad horizontal-pad]
     [vertical-pad vertical-pad]])
→ (is-a?/c close-icon%)
 parent : (is-a? area-container<%>)
 callback : (-> any) = void
 bg-color : (or/c #f string (is-a?/c color%)) = #f
 horizontal-pad : positive-integer? = 4
 vertical-pad : positive-integer? = 4
```

The *callback* is called when the close icon is clicked.

If *bg-color* is specified, it is used as the background color of the icon.

5 GIF and Animated GIF Writing

```
(require mrlib/gif)
```
package: gui-lib

```
(write-gif bitmap filename) → void?
  bitmap : (or/c (is-a?/c bitmap%)
                 (-> (is-a?/c bitmap%)))
  filename : path-string?
```

Writes the given *bitmap* to *filename* as a GIF image, where *bitmap* is either an instance of bitmap% or a thunk (to be called just once) that generates such an object. If the bitmap uses more than 256 colors, it is automatically quantized using a simple algorithm; see quantize. If the bitmap has a mask bitmap via get-loaded-mask, it is used to determine transparent pixels in the generated GIF image.

```
(write-animated-gif  bitmaps
                     delay-csec
                     filename
                    [#:loop? loop?
                     #:one-at-a-time? one-at-a-time?
                     #:last-frame-delay last-frame-delay])
 → void?
             (and/c
               (listof (or/c (is-a?/c bitmap%)
  bitmaps :                  (-> (is-a?/c bitmap%))))
               pair?)
  delay-csec : (integer-in 0 4294967295)
  filename : path-string?
  loop? : any/c = (and delay-csec #t)
  one-at-a-time? : any/c = #f
  last-frame-delay : (or/c (integer-in 0 4294967295) #f) = #f
```

Writes the bitmaps in *bitmaps* to *filename* as an animated GIF. The *bitmaps* list can contain a mixture of bitmap% objects and thunks (each called just once) that produce bitmap% objects. The *delay-csec* argument is the amount of time in 1/100s of a second to wait between transitions. If *loop?* is a true value, then the GIF is marked as a looping animation.

If *one-at-a-time?* is #f, then the content of all images is collected and quantized at once, to produce a single colortable; a drawback to this approach is that it uses more memory, and it allows less color variation among animation frames. Even when *one-at-a-time?* is #f, the result of each thunk in *bitmaps* is converted to a byte-string one at a time.

If *one-at-a-time?* is true, then the bitmaps are quantized and written to the file one at a time; that is, for each thunk in *bitmaps*, its result is written and discarded before another thunk is called. A drawback to this approach is that a separate colortable is written for each frame in the animation, which can make the resulting file large.

If `last-frame-delay` is not false, a delay of `last-frame-delay` (in 1/100s of a second) is added to the last frame. This extra delay is useful when `loop?` is true.

6 Graphs

```
(require mrlib/graph)
```
package: gui-lib

The mrlib/graph library provides a graph drawing toolkit built out of pasteboard%s.

6.1 graph-pasteboard<%>

```
graph-pasteboard<%> : interface?
```

```
(send a-graph-pasteboard get-arrowhead-params)
 → number number number
```

Returns the current settings for the arrowhead's drawing.

```
(send a-graph-pasteboard on-mouse-over-snips lst) → void?
  lst : (listof (is-a?/c snip%))
```

This method is called when the mouse passes over any snips in the editor. It is only called when the list of snips under the editor changes (ie, if the mouse moves, but remains over the same list of snips, the method is not called). Also, this method is called with the empty list if the mouse leaves the pasteboard.

```
(send a-graph-pasteboard set-arrowhead-params angle-width
                                              short-side
                                              long-size)
 → void?
  angle-width : real?
  short-side : real?
  long-size : real?
```

Sets drawing parameters for the arrowhead. The first is the angle of the arrowhead's point, in radians. The second is the length of the outside line of the arrowhead and the last is the distance from the arrowhead's point to the place where the arrowhead comes together.

```
(send a-graph-pasteboard set-draw-arrow-heads? draw-arrow-heads?)
 → void?
  draw-arrow-heads? : any/c
```

Sets a boolean controlling whether or not arrow heads are drawn on the edges between nodes.

This setting does not affect self-links—only links between two different nodes.

```
(send a-graph-pasteboard set-flip-labels? flip-labels?) → void?
  flip-labels? : any/c
```

Sets a boolean controlling whether or not arrow labels are flipped so the are always right-side-up. Note that if there are two nodes with edges going from the first to the second, and from the second to the first, and the two have labels, then this should be turned off or the labels will appear in the same space.

This setting does not affect self-links—only links between two different nodes.

```
(send a-graph-pasteboard set-edge-label-font font) → void?
  font : (is-a?/c font%)
```

Updates the font used to draw the edge labels.

```
(send a-graph-pasteboard get-edge-label-font)
 → (is-a?/c font%)
```

Returns the font currently being used to draw the edge labels

```
(send a-graph-pasteboard draw-edges dc
                                    left
                                    top
                                    right
                                    bottom
                                    dx
                                    dy)    → void?
  dc : (is-a?/c dc<%>)
  left : real?
  top : real?
  right : real?
  bottom : real?
  dx : real?
  dy : real?
```

This is called by the `on-paint` callback of a graph pasteboard, and is expected to draw the edges between the snips. The argments are a subset of those passed to `on-paint` and it is only called when the `before?` argument to `on-paint` is `#t`.

```
(send a-graph-pasteboard draw-single-edge dc
                                          dx
                                          dy
                                          from
                                          to
                                          from-x
                                          from-y
                                          to-x
                                          to-y
                                          arrow-point-ok?)
 → void?
  dc : (is-a?/c dc<%>)
  dx : real?
  dy : real?
  from : (is-a?/c graph-snip<%>)
  to : (is-a?/c graph-snip<%>)
  from-x : real?
  from-y : real?
  to-x : real?
  to-y : real?
  arrow-point-ok? : (-> real? real? boolean?)
```

This method is called to draw each edge in the graph, except for the edges that connect a node to itself.

The *dc*, *dx*, and *dy* arguments are the same as in on-paint.

The *from-x*, *from-y*, *to-x*, and *to-y* arguments specify points on the source and destination snip's bounding box where a straight line between the centers of the snip would intersect.

The *arrow-point-ok?* function returns #t when the point specified by its arguments is inside the smallest rectangle that covers both the source and destination snips, but is outside of both of the rectangles that surround the source and destination snips themselves.

This default implementation uses update-polygon to compute the arrowheads and otherwise draws a straight line between the two points and then the arrowheads, unless the arrowhead points are not ok according to *arrow-point-ok?*, in which case it just draws the line.

```
(send a-graph-pasteboard update-arrowhead-polygon from-x
                                                   from-y
                                                   to-x
                                                   to-y
                                                   point1
                                                   point2
                                                   point3
                                                   point4)
 → void?
  from-x : real?
  from-y : real?
  to-x : real?
  to-y : real?
  point1 : (is-a?/c point%)
  point2 : (is-a?/c point%)
  point3 : (is-a?/c point%)
  point4 : (is-a?/c point%)
```

Updates the arguments *point1*, *point2*, *point3*, *point4* with the coordinates of an arrowhead for a line that connects (*from-x*,*from-y*) to (*to-x*,*to-y*).

6.2 graph-pasteboard-mixin

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

```
(new graph-pasteboard-mixin
  [[edge-labels? edge-labels?]
   [edge-label-font edge-label-font]]
   [cache-arrow-drawing? cache-arrow-drawing?]
   ...superclass-args...)
→ (is-a?/c graph-pasteboard-mixin)
 edge-labels? : boolean? = #t
 edge-label-font : (or/c #f (is-a?/c font%)) = #f
 cache-arrow-drawing? : any
```

If *edge-labels?* is #f, no edge labels are drawn. Otherwise, they are.

If *edge-label-font* is supplied, it is used when drawing the labels on the edges. Otherwise, the font is not set before drawing the labels, defaulting to the dc<%> object's font.

If *cache-arrow-drawing?* is #f, then the arrows in the snip are not cached in a bitmap (to speed up drawing when the mouse moves around). Otherwise, they are.

This mixin overrides many methods to draw lines between graph-snip<%> that it contains.

6.3 graph-snip<%>

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

```
(send a-graph-snip add-child child) → void?
  child : (is-a?/c graph-snip<%>)
```

Adds a child of this snip. Instead of calling this method, consider using the add-links function.

```
(send a-graph-snip add-parent parent) → void?
  parent : (is-a?/c graph-snip<%>)
(send a-graph-snip add-parent parent
                              mouse-over-pen
                              mouse-off-pen
                              mouse-over-brush
                              mouse-off-brush) → void?
  parent : (is-a?/c graph-snip<%>)
  mouse-over-pen : (or/c false/c (is-a?/c pen%))
  mouse-off-pen : (or/c false/c (is-a?/c pen%))
  mouse-over-brush : (or/c false/c (is-a?/c brush%))
  mouse-off-brush : (or/c false/c (is-a?/c brush%))
```

Adds a parent of this snip. Instead of calling this method, consider using the add-links function.

```
(send a-graph-snip get-children) → (listof snip%)
```

returns a list of snips that implement graph-snip<%>. Each of these snips will have a line drawn from it, pointing at this snip.

```
(send a-graph-snip get-parents) → (listof graph-snip<%>)
```

Returns a list of snips that implement graph-snip<%>. Each of these snips will have a line drawn to it, starting from this snip.

```
(send a-graph-snip remove-child child) → void?
  child : (is-a?/c graph-snip<%>)
```

Removes a child snip from this snip. Be sure to remove this snip as a parent from the argument, too. Instead of calling this method, consider using the remove-links function.

```
(send a-graph-snip remove-parent parent) → void?
  parent : (is-a?/c graph-snip<%>)
```

Removes a parent snip from this snip. Be sure to remove this snip as a child from the argument, too. Instead of calling this method, consider using the `remove-links` function.

```
(send a-graph-snip set-parent-link-label parent
                                         label) → void?
  parent : (is-a?/c graph-snip<%>)
  label : (or/c false/c string/)
```

Changes the label on the edge going to the *parent* to be *label*. Ignored if no such egde exists.

6.4 graph-snip-mixin

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

6.5 Graph Functions

```
(add-links parent child) → void?
  parent : (is-a?/c graph-snip<%>)
  child : (is-a?/c graph-snip<%>)
(add-links parent child) → void?
  parent : (is-a?/c graph-snip<%>)
  child : (is-a?/c graph-snip<%>)
(add-links  parent
            child
            dark-pen
            light-pen
            dark-brush
            light-brush
           [label])      → void?
  parent : (is-a?/c graph-snip<%>)
  child : (is-a?/c graph-snip<%>)
  dark-pen : (or/c (is-a?/c pen) false/c)
  light-pen : (or/c (is-a?/c pen) false/c)
  dark-brush : (or/c (is-a?/c brush%) false/c)
  light-brush : (or/c (is-a?/c brush%) false/c)
```

```
  label : (or/c string? false/c) = #f
(add-links  parent
            child
            dark-pen
            light-pen
            dark-brush
            light-brush
            dx
            dy
           [label])      → void?
  parent : (is-a?/c graph-snip<%>)
  child : (is-a?/c graph-snip<%>)
  dark-pen : (or/c (is-a?/c pen) false/c)
  light-pen : (or/c (is-a?/c pen) false/c)
  dark-brush : (or/c (is-a?/c brush%) false/c)
  light-brush : (or/c (is-a?/c brush%) false/c)
  dx : real?
  dy : real?
  label : (or/c string? false/c) = #f
```

Connects a parent snip to a child snip within a pasteboard.

The default *dark-pen*/*dark-brush* and *light-pen*/*light-brush* are blue and purple, respectively. The *dark-pen* and *dark-brush* are used when the mouse cursor is over the snip (or a child or parent), and the *light-pen* and *light-brush* are used when the mouse cursor is not over the snip. The brush is used to draw inside the arrow head and the pen is used to draw the border of the arrowhead and the line connecting the two snips.

if *label* is provided and not `#f`, it is used as a label on the edge.

When *dx* and *dy* are provided, the are offsets for the head and the tail of the arrow. Otherwise, `0` offsets are used.

```
(add-links/text-colors parent
                       child
                       dark-pen
                       light-pen
                       dark-brush
                       light-brush
                       dark-text
                       light-text
                       dx
                       dy
                       label)      → void?
  parent : (is-a?/c graph-snip<%>)
  child : (is-a?/c graph-snip<%>)
```

```
  dark-pen : (or/c (is-a?/c pen) false/c)
  light-pen : (or/c (is-a?/c pen) false/c)
  dark-brush : (or/c (is-a?/c brush%) false/c)
  light-brush : (or/c (is-a?/c brush%) false/c)
  dark-text : (or/c (is-a?/c color%) false/c)
  light-text : (or/c (is-a?/c color) false/c)
  dx : real?
  dy : real?
  label : (or/c string? false/c)
```

Like add-links, but with extra *dark-text* and *light-text* arguments to set the colors of the label.

```
(remove-links parent child) → void?
  parent : (is-a?/c graph-snip<%>)
  child : (is-a?/c graph-snip<%>)
```

Disconnects a parent snip from a child snip within a pasteboard.

```
(set-link-label parent child label) → void?
  parent : (is-a?/c graph-snip<%>)
  child : (is-a?/c graph-snip<%>)
  label : (or/c string? false/c)
```

Changes the label on the edge going from *child* to *parent* to be *label*. If there is no existing edge between the two nodes, then nothing happens.

```
(dot-positioning  pb
                 [option
                  overlap-or-horizontal?]) → void?
  pb : (is-a?/c pasteboard%)
  option :  (or/c dot-label neato-label neato-hier-label neato-ipsep-label)
         = dot-label
  overlap-or-horizontal? : boolean? = #f
```

This finds the sizes of the graph-snip<%>s in *pb* and their children and then passes that information to `dot` or `neato` (depending on *option*), extracting a layout and then applying it to the snips in *pb*.

If *option* is dot-label, then *overlap-or-horizontal?* controls whether `dot` uses a horizontal or vertical alignment. If *option* is any of the other options, it controls whether or not `neato` is allowed to overlap nodes.

If find-dot returns #f, this function does nothing.

```
(find-dot [neato?]) → (or/c path? #f)
  neato? : boolean? = #f
```

Tries to find the `dot` or `neato` binary and, if it succeeds, returns the path to it. If it cannot find it, returns `#f`.

```
dot-label : string?
```

A string describing the regular dot option for graph layout that `dot-positioning` uses.

```
neato-label : string?
```

A string describing the neato option for graph layout that `dot-positioning` uses.

```
neato-hier-label : string?
```

A string describing the neato hierarchical option for graph layout that `dot-positioning` uses.

```
neato-ipsep-label : string?
```

A string describing the neato ipsep option for graph layout that `dot-positioning` uses.

7 Hierarchical List Control

```
(require mrlib/hierlist)
```
package: gui-lib

A hierarchical-list% control is a list of items, some of which can themselves be hierarchical lists. Each such sub-list has an arrow that the user can click to hide or show the sub-list's items.

The list control supports the following default keystrokes:

- Down: move to the next entry at the current level (skipping lower levels).
- Up: move to the previous entry at the current level (skipping lower levels).
- Left: move to the enclosing level (only valid at embedded levels).
- Right: move down in one level (only valid for lists).
- Return: open/close the current selected level (only valid for lists).

7.1 hierarchical-list%

```
hierarchical-list% : class?
  superclass: editor-canvas%
```

Creates a hierarchical-list control.

```
(new hierarchical-list%
   [parent parent]
  [[style style]])
→ (is-a?/c hierarchical-list%)
 parent :  (or/c (is-a?/c frame%) (is-a?/c dialog%)
                 (is-a?/c panel%) (is-a?/c pane%))
 style :   (listof (one-of/c 'no-border 'control-border 'combo
                             'no-hscroll 'no-vscroll
                             'hide-hscroll 'hide-vscroll
                             'auto-vscroll 'auto-hscroll
                             'resize-corner 'deleted 'transparent))
       = '(no-hscroll)
```

Creates the control.

If the style 'transparent is passed, then the use-style-background method will be called with #t when editor snips are created as part of the hierarchical list, ensuring that the entire control is transparent.

```
(send a-hierarchical-list get-selected)
 → (or/c (is-a?/c hierarchical-list-item<%>)
         false/c)
```

Returns the currently selected item, if any.

```
(send a-hierarchical-list new-item [mixin])
 → (is-a?/c hierarchical-list-item<%>)
          ((implementation?/c hierarchical-list-item<%>)
  mixin :   . -> .
            (implementation?/c hierarchical-list-item<%>))
        = (lambda (%) %)
```

Creates and returns a new (empty) item in the list. See `hierarchical-list-item<%>` for methods to fill in the item's label.

The *mixin* argument is applied to a class implementing `hierarchical-list-item<%>`, and the resulting class is instantiated as the list item.

```
(send a-hierarchical-list set-no-sublists no-sublists?) → void?
  no-sublists? : any/c
```

Enables/disables sublist mode. When sublists are disabled, space to the left of the list items (that would normally align non-list items with list items) is omitted. This method can be called only when the list is empty.

```
(send a-hierarchical-list new-list [mixin])
 → (is-a?/c hierarchical-list-compound-item<%>)
          ((implementation?/c hierarchical-list-compound-item<%>)
  mixin :   . -> .
            (implementation?/c hierarchical-list-compound-item<%>))
        = (lambda (%) %)
```

Creates and returns a new (empty) sub-list in the list. See `hierarchical-list-compound-item<%>` for methods to fill in the item's label and content.

The *mixin* argument is applied to a class implementing `hierarchical-list-compound-item<%>`, and the resulting class is instantiated as the sub-list.

```
(send a-hierarchical-list delete-item i) → void?
  i : (is-a?/c hierarchical-list-item<%>)
```

Deletes immediate item or sub-list *i* from the list.

```
(send a-hierarchical-list get-items)
 → (listof (is-a?/c hierarchical-list-item<%>))
```

Returns a list of all immediate items in the list control.

```
(send a-hierarchical-list selectable) → boolean?
(send a-hierarchical-list selectable on?) → void?
  on? : any/c
```

Reports whether items are selectable, or enables/disables item selection.

```
(send a-hierarchical-list on-select i) → any
  i : (or/c (is-a?/c hierarchical-list-item<%>) false/c)
```

Called for new select of `i`, where `i` is `#f` if no item is now selected.

```
(send a-hierarchical-list on-click i) → any
  i : (is-a?/c hierarchical-list-item<%>)
```

Called when an item is clicked on, but selection for that item is not allowed. Selection can be disallowed by `selectable` or `set-allow-selection` in `hierarchical-list-item<%>`.

```
(send a-hierarchical-list on-double-select i) → any
  i : (is-a?/c hierarchical-list-item<%>)
```

Called for a double-click on `i`.

```
(send a-hierarchical-list on-item-opened i) → any
  i : (is-a?/c hierarchical-list-compound-item<%>)
```

Called when the arrow for `i` is turned down.

```
(send a-hierarchical-list on-item-closed i) → any
  i : (is-a?/c hierarchical-list-compound-item<%>)
```

Called when the arrow for `i` is turned up.

```
(send a-hierarchical-list sort  less-than-proc
                                [recur?])       → void?
                    ((is-a?/c hierarchical-list-item<%>)
  less-than-proc :  (is-a?/c hierarchical-list-item<%>)
                     . -> . any/c)
  recur? : any/c = #t
```

Sorts items in the list by calling *less-than-proc* on pairs of items. If *recur?* is true, items in sub-lists are sorted recursively.

```
(send a-hierarchical-list can-do-edit-operation? op
                                                 [recursive?])
 → boolean?
  op : symbol?
  recursive? : any/c = #t
```

Like can-do-edit-operation? in editor<%>. The default implementation always returns #f.

```
(send a-hierarchical-list do-edit-operation op
                                           [recursive?]) → void?
  op : symbol?
  recursive? : any/c = #t
```

Like do-edit-operation in editor<%>. The default implementation does nothing.

```
(send a-hierarchical-list select-prev) → void?
(send a-hierarchical-list select-next) → void?
(send a-hierarchical-list select-first) → void?
(send a-hierarchical-list select-last) → void?
(send a-hierarchical-list select-in) → void?
(send a-hierarchical-list select-out) → void?
(send a-hierarchical-list page-up) → void?
(send a-hierarchical-list page-down) → void?
```

Move the selection, scroll, and call on-select.

```
(send a-hierarchical-list select i) → void?
  i : (or/c (is-a?/c hierarchical-list-item<%>) false/c)
```

Moves the selection, scrolls as necessary to show it, and calls on-select unless disabled via on-select-always.

The allow-deselect method controls whether *i* is allowed to be #f to deselect the currently selected item.

```
(send a-hierarchical-list click-select i) → void?
  i : (or/c (is-a?/c hierarchical-list-item<%>) false/c)
```

Like select, but always calls on-select.

```
(send a-hierarchical-list on-select-always) → boolean?
(send a-hierarchical-list on-select-always always?) → void?
  always? : any/c
```

Gets/sets whether the `on-select` method is called in response to `select` (as opposed to `click-select`).

The initial mode enables `on-select` calls always.

```
(send a-hierarchical-list on-click-always) → boolean?
(send a-hierarchical-list on-click-always always?) → void?
  always? : any/c
```

Gets/sets whether the `on-click` method is called in response to all mouse clicks (as opposed to only when selected). `on-click` is called before `on-select`, if it is called (if the click results in selction).

This is initially disabled, by default.

```
(send a-hierarchical-list allow-deselect) → boolean?
(send a-hierarchical-list allow-deselect allow?) → void?
  allow? : any/c
```

Gets/sets whether the `on-select` can be called with a `#f` argument to deselect the current item (leaving none selected).

The initial mode does not allow deselection.

7.2 hierarchical-list-item<%>

```
hierarchical-list-item<%> : interface?
```

Instantiate this interface via `new-item`.

```
(send a-hierarchical-list-item get-editor) → (is-a?/c text%)
```

Returns a text-editor buffer whose content is the display representation of the item. In other words, fill in this text editor to set the item's label.

```
(send a-hierarchical-list-item is-selected?) → boolean?
```

Reports whether the item is selected.

```
(send a-hierarchical-list-item select on?) → void?
  on? : any/c
(send a-hierarchical-list-item click-select on?) → void?
  on? : any/c
```

Calls `select` or `click-select`. The *on?* argument can be `#f` only if `allow-deselect` in `hierarchical-list%` allows it.

```
(send a-hierarchical-list-item user-data) → any/c
(send a-hierarchical-list-item user-data data) → void?
  data : any/c
```

Gets/sets arbitrary data associated with the item.

```
(send a-hierarchical-list-item get-clickable-snip)
 → (is-a?/c snip%)
```

Returns the snip that (when clicked) selects this element the list. This method is intended for use with an automatic test suite.

```
(send a-hierarchical-list-item get-allow-selection?)
 → boolean?
(send a-hierarchical-list-item set-allow-selection allow?)
 → void?
  allow? : any/c
```

Gets/sets whether this item is allowed to be selected.

```
(send a-hierarchical-list-item get-parent)
 → (or/c (is-a?/c hierarchical-list-compound-item<%>) #f)
```

Returns the compound list item that contains the item or `#f` if none exists.

7.3 hierarchical-list-compound-item<%>

```
hierarchical-list-compound-item<%> : interface?
  implements: hierarchical-list-item<%>
```

Instantiate this interface via `new-list`.

```
(send a-hierarchical-list-compound-item new-item [mixin])
 → (is-a?/c hierarchical-list-item<%>)
          ((implementation?/c hierarchical-list-item<%>)
  mixin :   . -> .
            (implementation?/c hierarchical-list-item<%>))
        = (lambda (%) %)
```

Like `new-item` in `hierarchical-list%`.

```
(send a-hierarchical-list-compound-item set-no-sublists no-sublists?)
 → void?
  no-sublists? : any/c
```

Like set-no-sublists in hierarchical-list%.

```
(send a-hierarchical-list-compound-item new-list [mixin])
 → (is-a?/c hierarchical-list-compound-item<%>)
          ((implementation?/c hierarchical-list-compound-item<%>)
  mixin :   . -> .
            (implementation?/c hierarchical-list-compound-item<%>))
        = (lambda (%) %)
```

Like new-list in hierarchical-list%.

```
(send a-hierarchical-list-compound-item delete-item i) → void?
  i : (is-a?/c hierarchical-list-item<%>)
```

Deletes immediate item or sub-list *i* from the sub-list.

```
(send a-hierarchical-list-compound-item get-items)
 → (listof (is-a?/c hierarchical-list-item<%>))
```

Returns a list of all immediate items in the sub-list.

```
(send a-hierarchical-list-compound-item open) → void?
(send a-hierarchical-list-compound-item close) → void?
(send a-hierarchical-list-compound-item toggle-open/closed)
 → void?
```

Shows or hides the items of this sub-list.

```
(send a-hierarchical-list-compound-item is-open?) → boolean?
```

Reports whether the items of this sub-list are visible.

```
(send a-hierarchical-list-compound-item get-arrow-snip)
 → (is-a?/c snip%)
```

Returns a snip that corresponds to the arrow to hide/show items of the sub-list. The result is intended for use by automatic test suites.

7.4 Snips in a hierarchical-list% Instance

The find-snip in text% method of the editor in a hierarchical-list% return instances of hierarchical-item-snip% and hierarchical-list-snip%.

```
hierarchical-item-snip% : class?
  superclass: editor-snip%
```

```
(send a-hierarchical-item-snip get-item)
 → (is-a?/c hierarchical-list-item<%>)
```

Returns the hierarchical-list-item<%> corresponding to the snip.

```
hierarchical-list-snip% : class?
  superclass: editor-snip%
```

```
(send a-hierarchical-list-snip get-item)
 → (is-a?/c hierarchical-list-compound-item<%>)
```

Returns the hierarchical-list-compound-item<%> corresponding to the snip.

```
(send a-hierarchical-list-snip get-content-buffer)
 → (is-a?/c text%)
```

Returns the text% that contains the sub-item snips.

8 Include Bitmap

```
(require mrlib/include-bitmap)
```
package: gui-lib

The include-bitmap form takes a filename containing a bitmap and "inlines" the bitmap into the program.

Historically, the advantage of inlining the bitmap is that a stand-alone executable can be created that contains the bitmap and does not refer to the original image file. The define-runtime-path form, however, now provides a better alternative.

```
(include-bitmap path-spec)
(include-bitmap path-spec type-expr)
```

The *path-spec* is the same as for include form. The *type-expr* should produce 'unknown, 'unknown/mask, etc., as for bitmap%, and the default is 'unknown/mask.

```
(include-bitmap/relative-to source path-spec)
(include-bitmap/relative-to source path-spec [type-expr])
```

Analogous to include-at/relative-to, though only a source is needed (no context).

9 Interactive Value Port

```
(require mrlib/interactive-value-port)
```
package: gui-lib

```
(set-interactive-display-handler port) → void?
  port : output-port?
```

Sets *port*'s display handler (via port-display-handler) so that when it encounters these values:

- exact, real, non-integral numbers
- syntax objects

it uses write-special to send snips to the port, instead of those values. Otherwise, it behaves like the default handler.

To show values embedded in lists and other compound object, it uses pretty-print.

```
(set-interactive-write-handler port) → void?
  port : output-port?
```

Like set-interactive-display-handler, but sets the port-write-handler.

```
(set-interactive-print-handler port) → void?
  port : output-port?
```

Like set-interactive-display-handler, but sets the port-print-handler.

10 Name Message

```
(require mrlib/name-message)
```
package: gui-lib

```
name-message% : class?
  superclass: canvas%
```

A name-message% control displays a filename that the user can click to show the filename's path and select one of the enclosing directories. Override the on-choose-directory method to handle the user's selection.

```
(new name-message% ...superclass-args...)
 → (is-a?/c name-message%)
```

Passes all arguments to super-init.

```
(send a-name-message on-choose-directory dir) → void?
  dir : path-string?
```

Called when one of the popup menu items is chosen. The argument is a represents the selected directory.

```
(send a-name-message on-event event) → void?
  event : (is-a?/c mouse-event%)
```

Overrides on-event in canvas<%>.

Handles the click by popping up a menu or message.

```
(send a-name-message on-paint) → void?
```

Overrides on-paint in canvas%.

Draws the control's current message.

```
(send a-name-message set-hidden? hidden?) → void?
  hidden? : any/c
```

Calling this method with #f causes the name message to become invisible and to stop responding to mouse movements.

Calling it with a true value restores its visibility and makes it respond to mouse movements again.

```
(send a-name-message set-message file-name?
                                 msg)       → void?
  file-name? : any/c
  msg : (if filename? path-string? string?)
```

Sets the label for the control.

If *file-name?* is #t, *msg* is treated like a pathname, and a click on the name-message control creates a popup menu to open a get-file dialog.

If *file-name?* is #f, *msg* is treated as a label string. Clicking on the name-message control pops up a dialog saying that there is no file name until the file is saved.

```
(send a-name-message set-short-title short-title?) → void?
  short-title? : boolean?
```

Sets the *short-title?* flag. The flag defaults to #f.

If the flag is #t, then the label for the control is simply the string "/". Otherwise, the label is determined by the set-message.

```
(send a-name-message fill-popup menu reset) → any
  menu : (is-a?/c popup-menu%)
  reset : (-> void?)
```

This method is called when the user clicks in the name message. Override it to fill in the menu items for the popup menu *menu*.

```
(send a-name-message get-background-color)
 → (or/c #f (is-a/c color%) string?)
```

The result of this method is used for the background color when redrawing the name message. If it is #f, the OS's default panel background is used.

```
(send a-name-message set-allow-shrinking width) → void?
  width : (or/c #f number?)
```

When this method receives a number, the name-message will then shrink (the number indicates the minimum width the name message will have).

If it receives false, the name message will not shrink and its minimum width will be the size required to display its current label.

By default, the name-message does not allow shrinking.

```
(calc-button-min-sizes dc str [font]) → real? real?
  dc : (is-a?/c dc<%>)
  str : string?
  font : (or/c #f (is-a?/c font%)) = #f
```

Calculates the minimum width and height of a button label (when drawn with draw-button-label). Returns two values: the width and height. The *dc* argument is used for sizing.

```
(draw-button-label dc
                   str
                   dx
                   dy
                   width
                   height
                   mouse-over?
                   grabbed?
                   font
                   background) → void?
  dc : (is-a?/c dc<%>)
  str : string?
  dx : real?
  dy : real?
  width : real?
  height : real?
  mouse-over? : boolean?
  grabbed? : boolean?
  font : (is-a?/c font%)
  background : (or/c (is-a?/c color%) string? #f)
```

Draws a button label like the one for the (define ...) and filename buttons in the top-left corner of the DrRacket frame. Use this function to draw similar buttons.

The basic idea is to create a canvas object whose on-paint method is overridden to call this function. The *dc* argument should be canvas's drawing context, and *str* should be the string to display on the button. The *width* and *height* arguments should be the width and height of the button, and the *dx* and *dy* arguments specify an offset into *dc* for the button. The *mouse-over?* argument should be true when the mouse is over the button, and the *grabbed?* argument should be true when the button has been pressed. The *font* and *background* arguments supply the font to use in drawing (possibly normal-control-font) and the background color to paint (if any).

See calc-button-min-sizes for help calculating the min sizes of the button.

```
(pad-xywh tx ty tw th) → number? number? (>=/c 0) (>=/c 0)
  tx : number?
  ty : number?
  tw : (>=/c 0)
  th : (>=/c 0)
```

Returns spacing information describing how draw-button-label draws. The inputs are the x and y coordinates where the text should appear and the width and height of the text, and the results are the x and y coordinates where the shape should be drawn and the width and height of the overall shape.

11 Path Dialog

```
(require mrlib/path-dialog)
```
package: gui-lib

```
path-dialog% : class?
  superclass: dialog%
```

The path-dialog% class implements a platform-independent file/directory dialog. The dialog is similar in functionality to the get-file, put-file, get-directory, and get-file-list procedures, but considerable extra functionality is available through the path-dialog% class.

```
(new path-dialog%
   [[label label]
    [message message]
    [parent parent]
    [directory directory]
    [filename filename]
    [put? put?]
    [dir? dir?]
    [existing? existing?]
    [new? new?]
    [multi? multi?]
    [can-mkdir? can-mkdir?]
    [filters filters]
    [show-file? show-file?]
    [show-dir? show-dir?]
    [ok? ok?]
    [guard guard]])
→ (is-a?/c path-dialog%)
 label : (or/c label-string? false/c) = #f
 message : (or/c label-string? false/c) = #f
 parent : (or/c (is-a?/c frame%) (is-a?/c dialog%) false/c)
        = #f
 directory : (or/c path-string? false/c) = #f
 filename : (or/c path-string? false/c) = #f
 put? : any/c = #f
 dir? : any/c = #f
 existing? : any/c = (not put?)
 new? : any/c = #f
 multi? : any/c = #f
 can-mkdir? : any/c = put?
 filters : (or/c (listof (list string? string?))
                 (one-of/c #f #t))
         = #t
 show-file? : (or/c (path? . -> . any) false/c) = #f
```

```
show-dir? : (or/c (path? . -> . any) false/c) = #f
ok? : (or/c (path? . -> . any) false/c) = #f
guard : (or/c (path? . -> . any) false/c) = #f
```

The *label* argument is the dialog's title string. If *label* is #f, the default is based on other field values.

The *message* argument is a prompt message to show at the top of the dialog. If it is #f, no prompt line.

The *parent* argument is the parent frame or dialog, if any, for this dialog.

The *directory* argument specifies the dialog's initial directory. If it is #f, the initial directory is the last directory that was used by the user (or the current directory on first use).

The *filename* argument provides an initial filename text, if any.

If *put?* is true, the dialog operates in choose-file-to-write mode (and warn the user if choosing an existing name).

If *dir?* is true, the dialog operates in directory-choice mode.

If *existing?* is true, the use must choose an existing file.

If *new?* is true, the user must choose a non-existant path. Providing both *new?* and *existing?* as true triggers an exception.

If *multi?* is true, the dialog allows selection of multiple paths.

If *can-mkdir?* is true, the dialog includes a button for the user to create a new directory.

The *filters* argument is one of:

- (list (list *filter-name filter-glob*) ...) — a list of pattern names (e.g., "Racket Files") and glob patterns (e.g., "*.rkt;*.scrbl"). Any list, including an empty list, enables a filter box for the user to enter glob patterns, and the given list of choices is available in a combo-box drop-down menu. Glob patterns are the usual Unix ones (see glob->regexp), and a semicolon can be used to allow multiple patterns.
- #f — no patterns and no filter input box.
- #t — use a generic "All" filter, which is "*.*" on Windows and "*" on other platforms.

The *show-file?* predicate is used to filter file paths that are shown in the dialog. The predicate is applied to the file name as a string while the current-directory parameter is set. This predicate is intended to be a lightweight filter for choosing which names to display.

The *show-dir?* predicate is similar, but for directories instead of files.

The *ok?* predicate is used in a similar fashion to the *show-file?* and *show-dir?* predicate, but it is used to determine whether the OK button should be

enabled when a file or directory is selected (so it need not be as lightweight as the other predicates).

The *guard* procedure is a generic verifier for the dialog's final result, as produced by the run method. It receives the result that is about to be returned (which can be a list in a multi-selection dialog), and can return a different value (any value) instead. If it throws an exception, an error dialog is shown, and the dialog interaction continues (so it can be used to verify results without dismissing the dialog). This procedure can also raise #<void>, in which case the dialog remains without an error message.

(send *a-path-dialog* run) → any/c

Shows the dialog and returns the selected result. If a guard procedure is not supplied when the dialog is created, then the result is either a path or a list of paths (and the latter only when *multi?* is true when the dialog is created). If a *guard* procedure is supplied, its result determines the result of this method.

12 Plot

```
(require mrlib/plot)
```
package: gui-lib

The mrlib/plot library provides a simple tool for plotting data values to a device context.

This is an old library, kept only for compatibility. You will undoubtedly want to use the plot library instead, which offers many more features and is actively maintained.

```
(struct data-set (points connected? pen min-x max-x min-y max-y)
    #:extra-constructor-name make-data-set)
  points : (listof (is-a?/c point%))
  connected? : any/c
  pen : (is-a?/c pen%)
  min-x : real?
  max-x : real?
  min-y : real?
  max-y : real?
```

The points field contains the data values to plot, and connected? indicates whether the points are connected by a line. The pen field provides a pen for plotting points/lines. The remaining fields determine the plotting area within a drawing context.

```
(struct plot-setup (axis-label-font
                    axis-number-font
                    axis-pen
                    grid?
                    grid-pen
                    x-axis-marking
                    y-axis-marking
                    x-axis-label
                    y-axis-label)
    #:extra-constructor-name make-plot-setup)
  axis-label-font : (is-a?/c font%)
  axis-number-font : (is-a?/c font%)
  axis-pen : (is-a?/c pen)
  grid? : any/c
  grid-pen : (is-a?/c pen)
  x-axis-marking : (listof real?)
  y-axis-marking : (listof real?)
  x-axis-label : string?
  y-axis-label : string?
```

Configures a plot. The grid? field determines whether to draw a grid at axis markings, and the x-axis-marking and y-axis-marking lists supply locations for marks on each axis. The other fields are self-explanatory.

```
(plot dc data setup) → void?
  dc : (is-a?/c dc<%>)
  data : (listof data-set?)
  setup : plot-setup?
```

Draws the data-sets in *data* into the given *dc*. Uses drawing-context coordinates in data-sets that will accommodate all of the data sets.

13 Switchable Button

```
(require mrlib/switchable-button)
```
package: gui-lib

```
switchable-button% : class?
  superclass: canvas%
```

A switchable-button% control displays and icon and a string label. It toggles between display of just the icon and a display with the label and the icon side-by-side.

The panel:discrete-sizes-mixin explicitly uses switchable-button%s via their get-small-width, get-large-width, and get-without-label-small-width methods. See panel:discrete-sizes-mixin for more details.

```
(new switchable-button%
     [label label]
     [bitmap bitmap]
     [callback callback]
    [[alternate-bitmap alternate-bitmap]
     [vertical-tight? vertical-tight?]
     [min-width-includes-label? min-width-includes-label?]]
     ...superclass-args...)
 → (is-a?/c switchable-button%)
  label : (or/c string? (is-a?/c bitmap%) #f)
  bitmap : (is-a?/c bitmap%)
  callback : (-> (is-a?/c switchable-button%) any/c)
  alternate-bitmap : (is-a?/c bitmap%) = bitmap
  vertical-tight? : boolean? = #f
  min-width-includes-label? : boolean? = #f
```

The *callback* is called when the button is pressed. The *label* and *bitmap* are used as discussed above.

If *alternate-bitmap* is supplied, then it is used when the label is not visible (via a call to set-label-visible). If it is not supplied, both modes show *bitmap*.

If the *vertical-tight?* argument is #t, then the button takes up as little as possible vertical space.

If the *min-width-includes-label?* is #t, then the minimum width includes both the bitmap and the label. Otherwise, it includes only the bitmap.

```
(send a-switchable-button set-label-
visible visible?) → void?
  visible? : boolean?
```

Sets the visibility of the string part of the label.

```
(send a-switchable-button command) → void?
```

Calls the button's callback function.

```
(send a-switchable-button get-button-label) → string?
```

Returns the label of this button.

```
(send a-switchable-button get-large-width)
 → exact-nonnegative-integer?
```

Returns the width of the button when it would show both the label and the bitmap and when it is in label-visible mode (i.e., when `set-label-visible` has been called with `#t`).

```
(send a-switchable-button get-small-width)
 → exact-nonnegative-integer?
```

Returns the width of the button when it would show both just the bitmap (not the alternate bitmap), and when it is in label-visible mode (i.e., when `set-label-visible` has been called with `#t`).

```
(send a-switchable-button get-without-label-small-width)
 → exact-nonnegative-integer?
```

Returns the width of the button when it is not in label-visible mode (i.e., when `set-label-visible` has been called with `#f`).

14 Image Core

```
(require mrlib/image-core)
```
package: gui-lib

This library is the core part of the 2htdp/image library that DrRacket links into the namespace of all languages that it runs. This ensures that minimal support for these images are the same in all languages, specifically including support for printing the images and constructing the core data structures making up an image.

```
(render-image image dc dx dy) → void?
  image : image?
  dc : (is-a?/c dc<%>)
  dx : real?
  dy : real?
```

Draws *image* in *dc* at the position (*dx*,*dy*).

```
(image? v) → boolean?
  v : any/c
```

Recognizes the images that library handles.

```
(un/cache-image image b) → image?
  image : image?
  b : any/c
```

Returns an image that either caches its drawing in the snip draw method or doesn't, depending on *b*.

Not all image? values have special caching capabilities; in those cases, this returns a copy of the value if it is a snip%; otherwise it returns the value itself (if it isn't a snip%).

```
(compute-image-cache image) → void?
  image : image?
```

When the image has a bitmap-cache (which it does by default, although un/cache-image can disable it), this function fills in the bitmap, doing the work to draw image into the bitmap.

Ordinarily, the image's bitmap cache is computed the first time the image is actually rendered.

15 Matrix Snip

```
(require mrlib/matrix-snip)
```
package: gui-lib

The mrlib/matrix-snip library implements a matrix value that displays in 2-D graphical form.

```
visible-matrix% : class?
  superclass: cache-image-snip%
```

A 2-D graphical matrix.

16 TeX Table

```
(require mrlib/tex-table)
```
package: tex-table

```
tex-shortcut-table : (listof
                      (list/c string?
                              (lambda (x)
                                (and (string? x)
                                     (= (string-length x)
                                        1)))))
```

This is an association list mapping the shortcut strings that DrRacket uses with its `control-\` (or `command-\`) strings to their corresponding unicode characters. For example, it contains this mapping:

```
("alpha" "α")
```

as well as many more.

17 Terminal Window

```
(require mrlib/terminal)
```
package: gui-lib

The mrlib/terminal library provides a simple GUI wrapper around functions that normally would run in command-line scripts.

```
(in-terminal doit
             [#:container container
              #:cleanup-thunk cleanup-thunk
              #:title title
              #:abort-label abort-label
              #:aborted-message aborted-message
              #:canvas-min-width canvas-min-width
              #:canvas-min-height canvas-min-height
              #:close-button? close-button?
              #:close-label close-label])
 → (is-a?/c terminal<%>)
        (-> eventspace?
 doit :     (or/c (is-a?/c top-level-window<%>) #f)
            void?)
 container : (or/c #f (is-a?/c area-container<%>)) = #f
 cleanup-thunk : (-> void?) = void
 title : string? = "mrlib/terminal"
 abort-label : string?
             = (string-constant plt-installer-abort-installation)
 aborted-message : string?
                 = (string-constant plt-installer-aborted)
 canvas-min-width : (or/c #f (integer-in 0 10000)) = #f
 canvas-min-height : (or/c #f (integer-in 0 10000)) = #f
 close-button? : boolean? = #t
 close-label : string? = (string-constant close)
```

Creates a GUI, sets up the current error and output ports to print into the GUI's content, and calls *doit* in a separate thread under a separate custodian. The exit-handler is set to a function that shuts down the new custodian.

The GUI is created in a new frame%, unless *container* is not #f, in which case the GUI is created as a new panel inside *container*. If a frame is created, it is provided as the second argument to *doit*, otherwise the second argument to *doit* is #f. If a frame is created, its title is *title*.

The result of in-terminal is a terminal<%> object that reports on the state of the terminal; this result is produced just after *doit* is started.

The *cleanup-thunk* is called on a queued callback to the eventspace active when with-

`installer-window` is invoked after *`doit`* completes.

In addition to the I/O generated by *`doit`*, the generated GUI contains two buttons: the abort button (with label *`abort-label`*) and the close button (with label *`close-label`*). The close button is present only if *`close-button?`* is `#t`.

When the abort button is pushed, the newly created custodian is shut down and the *`aborted-message`* is printed in the dialog. The close button becomes active when *`doit`* returns or when the thread running it is killed (via a custodian shut down, typically).

If *`container`* is `#f`, then the close button closes the frame; otherwise, the close button causes the container created for the terminal's GUI to be removed from its parent.

The *`canvas-min-width`* and *`canvas-min-height`* are passed to the *`min-width`* and *`min-height`* initialization arguments of the `editor-canvas%` object that holds the output generated by *`doit`*.

The value of `on-terminal-run` is invoked after *`doit`* returns, but not if it is aborted or an exception is raised.

```
(on-terminal-run) → (-> void?)
(on-terminal-run run) → void?
  run : (-> void?)
```

Invoked by `in-terminal`.

```
terminal<%> : interface?
```

The interface of a terminal status and control object produced by `in-terminal`.

```
(send a-terminal is-closed?) → boolean?
```

Returns `#t` if the terminal GUI has been closed, `#f` otherwise.

```
(send a-terminal close) → void?
```

Closes the terminal GUI. Call this method only if `can-close?` returns `#t`.

```
(send a-terminal can-close?) → boolean?
```

Reports whether the terminal GUI can be closed, because the terminal process is complete (or, equivalently, whether the close button is enabled).

```
(send a-terminal can-close-evt) → evt?
```

Returns a synchronizable event that becomes ready for synchronization when the terminal GUI can be closed.

```
(send a-terminal get-button-panel)
 → (is-a?/c horizontal-panel%)
```

Returns a panel that contains the abort and close buttons.

18 Acknowledgments

Contributors to this set of libraries include Mike MacHenry.

Index