PLT MrLib: Graphical Libraries Manual

PLT (scheme@plt-scheme.org)

301
Released December 2006

Copyright notice

Send us your Web links

If you use any parts or all of the PLT Scheme package (software, lecture notes) for one of your courses, for your research, or for your work, we would like to know about it. Furthermore, if you use it and publicize the fact on some Web page, we would like to link to that page. Please drop us a line at *scheme@plt-scheme.org*. Evidence of interest helps the DrScheme Project to maintain the necessary intellectual and financial support. We appreciate your help.

Thanks

Contributors to MrLib include Mike MacHenry.

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

This manual was typeset on January 12, 2006.

Contents

1. MrLib

The MrLib collection consists of several libraries, each of which provides a set of procedures and syntax.

To use a MrLib library, either at the top-level or within a module, import it with

```
(require (lib "libname" "mrlib"))
```

For example, to use the **graph.ss** library:

```
(require (lib "graph.ss" "mrlib"))
```

The MrLib collection provides the following libraries:

- **aligned-pasteboard.ss** – pasteboards that manage the geometry of their snips
- **bitmap-label.ss** – builds button labels from bitmaps
- **cache-image-snip.ss** – the core of the drscheme image teachpack
- **fancy-value-port.ss** – allows ports to show syntax and numbers interactively
- **graph.ss** – shows graphs in a pasteboard
- **include-bitmap.ss** – inlines bitmaps into source text for stand-alone executables
- **name-message.ss** – a control for showing a path in a pop-down menu
- **plot.ss** – library for plotting values to a dc
- **syntax-browser.ss** – library for browsing the structure of syntax objects
- **text-string-style-desc.ss** – normalizes snips and styles into sexps

2. aligned-pasteboard.ss: Aligned Pasteboard

To load: `(require (lib "aligned-pasteboard.ss" "mrlib"))`

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

- `aligned-editor-canvas%`
- `aligned-editor-snip%`
- `aligned-pasteboard<%>`
- `aligned-pasteboard-parent<%>`
- `horizontal-pasteboard%`
- `stretchable-snip<%>`
- `vertical-pasteboard%`

2.1 aligned-editor-canvas%

Superclass: `editor-canvas%`

Calls the `realign` method when resized.

```
- (new aligned-editor-canvas% (parent _) [(editor _)] [(style _)] [(scrolls-per-page
  _)] [(label _)] [(wheel-step _)] [(line-count _)] [(horizontal-inset _)] [(vertical-inse
  _)] [(enabled _)] [(vert-margin _)] [(horiz-margin _)] [(min-width _)] [(min-height
  _)] [(stretchable-width _)] [(stretchable-height _)]) ⇒ aligned-editor-canvas%
  object
    parent : frame%, dialog%, panel%, or pane% object
    editor = #f : text% or pasteboard% object or #f
    style = null : list of symbols in '(no-border control-border combo no-hscroll
                   no-vscroll hide-hscroll hide-vscroll auto-vscroll
                   auto-hscroll resize-corner deleted transparent)
    scrolls-per-page = 100 : exact integer in [1, 10000]
    label = #f : string (up to 200 characters) or #f
    wheel-step = 3 : exact integer in [1, 10000] or #f
    line-count = #f : exact integer in [1, 1000] or #f
    horizontal-inset = 5 : exact integer in [0, 1000]
    vertical-inset = 5 : exact integer in [0, 1000]
    enabled = #t : boolean
    vert-margin = 0 : exact integer in [0, 1000]
    horiz-margin = 0 : exact integer in [0, 1000]
```

min-width = 0 : exact integer in [0, 10000]
min-height = 0 : exact integer in [0, 10000]
stretchable-width = #t : boolean
stretchable-height = #t : boolean

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

The *style* list can contain the following flags:

- 'no-border — omits a border around the canvas
- 'control-border — gives the canvas a border that is like a text-field% control
- 'combo — gives the canvas a combo button that is like a combo-field% control; this style is intended for use with 'control-border, 'hide-hscroll, and 'hide-vscroll
- 'no-hscroll — disallows horizontal scrolling and hides the horizontal scrollbar
- 'no-vscroll — disallows vertical scrolling and hides the vertical scrollbar
- 'hide-hscroll — allows horizontal scrolling, but hides the horizontal scrollbar
- 'hide-vscroll — allows vertical scrolling, but hides the vertical scrollbar
- 'auto-hscroll — automatically hides the horizontal scrollbar when unneeded (unless 'no-hscroll or 'hide-hscroll is specified)
- 'auto-vscroll — automatically hides the vertical scrollbar when unneeded (unless 'no-vscroll or 'hide-vscroll is specified)
- 'resize-corner — leaves room for a resize control at the canvas's bottom right when only one scrollbar is visible
- 'deleted — creates the canvas as initially hidden and without affecting *parent*'s geometry; the canvas can be made active later by calling *parent*'s add-child method
- 'transparent — the canvas is "erased" before an update using it's parent window's background

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

If provided, the *wheel-step* argument is passed on to the wheel-step method. The default wheel step can be overridden globally though the '|MrEd:wheelStep| preference; see "Preferences" (§ in *PLT MrEd: Graphical Toolbox Manual*).

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

If *horizontal-inset* is not 5, it is passed on to the horizontal-inset method. Similarly, if *vertical-inset* is not 5, it is passed on to the vertical-inset method.

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

2.2 aligned-editor-snip%

Superclass: editor-snip%

Calls the realign method when resized.

- (new aligned-editor-snip% [(editor _)] [(with-border? _)] [(left-margin _)] [(top-margin _)] [(right-margin _)] [(bottom-margin _)] [(left-inset _)] [(top-inset _)] [(right-inset _)] [(bottom-inset _)] [(min-width _)] [(max-width _)] [(min-height _)] [(max-height _)]) ⇒ aligned-editor-snip% object
  editor = #f : text% object or #f
  with-border? = #t : boolean
  left-margin = 5 : exact non-negative integer
  top-margin = 5 : exact non-negative integer
  right-margin = 5 : exact non-negative integer

bottom-margin = 5 : exact non-negative integer
left-inset = 1 : exact non-negative integer
top-inset = 1 : exact non-negative integer
right-inset = 1 : exact non-negative integer
bottom-inset = 1 : exact non-negative integer
min-width = 'none : non-negative real number or 'none
max-width = 'none : non-negative real number or 'none
min-height = 'none : non-negative real number or 'none
max-height = 'none : non-negative real number or 'none

If *editor* is non-#f, then it will be used as the editor contained by the snip. See also set-editor.

If *with-border?* is not #f, then a border will be drawn around the snip. The editor display will be inset in the snip area by the amounts specified in the *-margin* arguments. The border will be drawn with an inset specified by the *-inset* arguments.

See get-inset and get-margin for information about the inset and margin arguments.

2.3 aligned-pasteboard<%>

get-aligned-min-height

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-height) ⇒ number

get-aligned-min-width

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

- (send *an-aligned-pasteboard* get-aligned-min-width) ⇒ number

realign

- (send *an-aligned-pasteboard* realign *width height*) ⇒ void
  width : nonnegative
  height : nonnegative

  Realigns the children inside the aligned-pasteboard<%> to the given width and height.
- (send *an-aligned-pasteboard* realign) ⇒ void

  Realigns the children inside the aligned-pasteboard¡the previously alloted width and height.

set-aligned-min-sizes

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.

- (send *an-aligned-pasteboard* set-aligned-min-sizes) ⇒ number

2.4 aligned-pasteboard-parent<%>

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

set-aligned-min-sizes

- (send *an-aligned-pasteboard-parent* set-aligned-min-sizes) ⇒ void

2.5 horizontal-pasteboard%

Superclass: pasteboard%

Implements: aligned-pasteboard<%>

- (new horizontal-pasteboard%) ⇒ horizontal-pasteboard% object

  The editor will not be displayed until it is attached to an editor-canvas% object or some other display.

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

after-delete

Called after a snip is deleted from the editor (and after the display is refreshed; use on-delete and begin-edit-sequence to avoid extra refreshes when after-delete modifies the editor).

See also can-delete? and on-edit-sequence.

No internals locks are set when this method is called.

- (send *an-horizontal-pasteboard* after-delete *snip*) ⇒ void
  snip : snip% object

after-insert

Called after a snip is inserted into the editor (and after the display is refreshed; use on-insert and begin-edit-sequence to avoid extra refreshes when after-insert modifies the editor).

See also can-insert? and on-edit-sequence.

No internals locks are set when this method is called.

- (send *an-horizontal-pasteboard* after-insert *snip before x y*) ⇒ void
  snip : snip% object
  before : snip% object or #f
  x : real number
  y : real number

after-reorder

Called before a snip is moved in the pasteboard's front-to-back snip order (and after the display is refreshed; use on-reorder and begin-edit-sequence to avoid extra refreshes when after-reorder modifies the editor).

See also can-reorder? and on-edit-sequence.

No internals locks are set when this method is called.

- (send *an-horizontal-pasteboard* after-reorder *snip to-snip before?*) ⇒ void
  snip : snip% object
  to-snip : snip% object
  before? : boolean

resized

Called (indirectly) by snips within the editor: it forces a recalculation of the display information in which the specified snip has changed its size.

- (send *an-horizontal-pasteboard* resized *snip redraw-now?*) ⇒ void
  snip : snip% object
  redraw-now? : boolean

2.6 stretchable-snip<%>

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

get-aligned-min-height

The minimum height that the snip can be resized to

- (send *a-stretchable-snip* get-aligned-min-height) ⇒ number

get-aligned-min-width

The minimum width that the snip can be resized to.

- (send *a-stretchable-snip* get-aligned-min-width) ⇒ number

stretchable-height

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

- (send *a-stretchable-snip* stretchable-height) ⇒ boolean

stretchable-width

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

- (send *a-stretchable-snip* stretchable-width) ⇒ boolean

2.7 vertical-pasteboard%

Superclass: pasteboard%

Implements: aligned-pasteboard<%>

- (new vertical-pasteboard%) ⇒ vertical-pasteboard% object

  The editor will not be displayed until it is attached to an editor-canvas% object or some other display.

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

after-delete

Called after a snip is deleted from the editor (and after the display is refreshed; use on-delete and begin-edit-sequence to avoid extra refreshes when after-delete modifies the editor).

See also can-delete? and on-edit-sequence.

No internals locks are set when this method is called.

- (send *a-vertical-pasteboard* after-delete *snip*) ⇒ void
  snip : snip% object

after-insert

Called after a snip is inserted into the editor (and after the display is refreshed; use on-insert and begin-edit-sequence to avoid extra refreshes when after-insert modifies the editor).

See also can-insert? and on-edit-sequence.

No internals locks are set when this method is called.

- (send *a-vertical-pasteboard* after-insert *snip before x y*) ⇒ void
  snip : snip% object
  before : snip% object or #f
  x : real number
  y : real number

after-reorder

Called before a snip is moved in the pasteboard's front-to-back snip order (and after the display is refreshed; use on-reorder and begin-edit-sequence to avoid extra refreshes when after-reorder modifies the editor).

See also can-reorder? and on-edit-sequence.

No internals locks are set when this method is called.

- (send *a-vertical-pasteboard* after-reorder *snip to-snip before?*) ⇒ void
  snip : snip% object

to-snip : snip% object
before? : boolean

resized

Called (indirectly) by snips within the editor: it forces a recalculation of the display information in which the specified snip has changed its size.

- (send *a-vertical-pasteboard* resized *snip redraw-now?*) ⇒ void
  snip : snip% object
  redraw-now? : boolean

3. bitmap-label.ss: Bitmap Labels

To load: `(require (lib "bitmap-label.ss" "mrlib"))`

`(make-bitmap-label` *`string bitmap-or-path`* [*`font`*]`)` PROCEDURE

Constructs a bitmap label suitable for use a button that contains the image named by *`bitmap-or-path`* followed by the text in *`string`*.

`(bitmap-label-maker` *`string bitmap-or-path`*`)` PROCEDURE

OBSOLETE. Use `make-bitmap-label` instead.

This function takes a string and `bitmap%` (or filename) and returns a function that takes a container and generates a bitmap. The container determines the font that is used for the bitmap label. The result is a bitmap (with a mask) that is suitable for use as a `button%` label.

4. cache-image-snip.ss: Cache-image Snip

To load: `(require (lib "cache-image-snip.ss" "mrlib"))`

This library is the core data structure for the image.ss teachpack. Images in the image.ss library 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.

4.1 `cache-image-snip%`

Superclass: `snip%`

- `(make-object cache-image-snip%)` ⇒ `cache-image-snip%` object

  Creates a plain snip of length 1 with the `"Basic"` style of `the-style-list`.

`get-argb`

- `(send` *`a-cache-image-snip`* `get-argb)` ⇒ argb

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

`get-argb-proc`

- `(send` *`a-cache-image-snip`* `get-argb-proc)` ⇒ (argb number number . -¿ . void)

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

`get-argb/no-compute`

- `(send` *`a-cache-image-snip`* `get-argb/no-compute)` ⇒ (union `#f` argb)

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

`get-bitmap`

- `(send` *`a-cache-image-snip`* `get-bitmap)` ⇒ bitmap

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

`get-dc-proc`

- `(send a-cache-image-snip get-dc-proc)` ⇒ (union #f ((is-a?/c dc<%>) int[dx] int[dy] -¿ void))

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

`get-pinhole`

- `(send a-cache-image-snip get-pinhole)` ⇒ (values number number)

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

`get-size`

- `(send a-cache-image-snip get-size)` ⇒ (values number number)

returns the width and height for the image.

4.2 Functions

snip-class SNIPCLASS

This snipclass is used for saved cache image snips.

`(make-argb` *`(vectorof rational[between 0 & 255]) int`*`)` CONSTRUCTOR

Constructs a new argb value. The vector has four entries for each pixel, an alpha, red, green, and blue value. The int specifies the width of the image; the height is the size of the vector, divided by 4, divided by the width.

`(argb-vector` *`argb`*`)` SELECTOR

Extracts the vector from an argb

`(argb-width` *`argb`*`)` SELECTOR

Extracts the width from an argb.

`(argb?` *`any`*`)` PREDICATE

Tests if its argument is an argb.

`(overlay-bitmap` *`dx dy color-bitmap mask-bitmap`*`)` PROCEDURE

Builds a new argb after overlaying color-bitmap (with masking based on mask-bitmap) onto argb at (dx, dy) from the top-left.

`(build-bitmap` *`draw w h`*`)` PROCEDURE

Builds a bitmap with width *`w`* and height *`h`*, using the procedure *`draw`*. The procedure should accept one argument, `dc<%>`, and return void.

`(flatten-bitmap` *`bitmap`*`)` PROCEDURE

Builds a new bitmap that flattens the original bitmap with its mask, producing a bitmap that has no mask, and looks the way that bitmap would draw (when drawn with the mask).

`(argb->cache-image-snip` *`argb number number`*`)` PROCEDURE

Builds a new `cache-image-snip%` based on the contents of argb, using the two numbers as the pinhole.

`(argb->bitmap` *`argb`*`)` PROCEDURE

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

5. interactive-value-port.ss: Interactive Value Port

To load: `(require (lib "interactive-value-port.ss" "mrlib"))`

`(set-interactive-display-handler` *`port`*`)` PROCEDURE

Sets the port-display-handler for the given port 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 port-display-handler.

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

`(set-interactive-write-handler` *`port`*`)` PROCEDURE

Same as *`set-interactive-display-handler`*, except that it sets the port-print-handler.

`(set-interactive-print-handler` *`port`*`)` PROCEDURE

Same as *`set-interactive-display-handler`*, except that it sets the port-display-handler.

6. graph.ss: Graph

To load: `(require (lib "graph.ss" "mrlib"))`

The graph.ss library provides a graph drawing toolkit, built out of MrEd's `pasteboard%`s.

- `graph-pasteboard<%>`
- `graph-pasteboard-mixin%`
- `graph-snip<%>`
- `graph-snip-mixin%`
- Graph Functions

6.1 `graph-pasteboard<%>`

`get-arrowhead-params`

- `(send` *`a-graph-pasteboard`* `get-arrowhead-params)` ⇒ (values number number number)

  Returns the current settings for the arrowhead's drawing.

`on-mouse-over`

- `(send` *`a-graph-pasteboard`* `on-mouse-over` *`a-los`*`)` ⇒ void
  `a-los` : (listof `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.

`set-arrowhead-params`

- `(send` *`a-graph-pasteboard`* `set-arrowhead-params` *`angle-width short-side long-size`*`)` ⇒ void
  `angle-width` : number
  `short-side` : number
  `long-size` : number

  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.

6.2 graph-pasteboard-mixin

Domain: (class->interface pasteboard%)

Implements: graph-pasteboard<%>

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

- init args:

  The editor will not be displayed until it is attached to an editor-canvas% object or some other display.

  A new keymap% object is created for the new editor. See also get-keymap and set-keymap.

  A new style-list% object is created for the new editor. See also get-style-list and set-style-list.

6.3 graph-snip<%>

add-child

- (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 provided from this library.

add-parent

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

- (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 : (union false? (is-a?/c pen%))
  mouse-off-pen : (union false? (is-a?/c pen%))
  mouse-over-brush : (union false? (is-a?/c brush%))
  mouse-off-brush : (union false? (is-a?/c brush%)))

get-children

- (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.

get-parents

- (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.

`remove-child`

- `(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.

`remove-parent`

- `(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.

6.4 `graph-snip-mixin`

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

Implements: `graph-snip<%>`

- `init args:`

Creates a plain snip of length 1 with the `"Basic"` style of `the-style-list`.

6.5 Graph Functions

`(add-links graph-snip<%> graph-snip<%>)` PROCEDURE

`(add-links graph-snip<%> graph-snip<%> (union false? (is-a?/c pen%)) (union false? (is-a?/c pen%)) (union false? (is-a?/c brush%)) (union false? (is-a?/c brush%))))` PROCEDURE

`(add-links graph-snip<%> graph-snip<%> (union false? (is-a?/c pen%)) (union false? (is-a?/c pen%)) (union false? (is-a?/c brush%)) (union false? (is-a?/c brush%)) number number)` PROCEDURE

The add-links function connects a parent snip to a child snip in the same pasteboard.

When called with two arguments, connects the snips using a blue/purple color scheme for the links. The first snip is the parent and the second snip is the child.

When called with four arguments, uses the two pens and brushes for the color scheme. The first pen and the first brush are used when the mouse cursor is over the snip (or a child or parent) and the second pen and 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.

The final arguments are dx and dy offsets for the head and the tail of the arrow, first dx and then dy.

7. include-bitmap.ss: Include Bitmap

To load: `(require (lib "include-bitmap.ss" "mrlib"))`

The **include-bitmap.ss** library provides a `include-bitmap` form that takes a filename containing a bitmap and "inlines" the bitmap into the program. 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.

`(include-bitmap` *`file-spec [type-expr]`*`)` SYNTAX

The *`file-spec`* is the same as for MzLib's `include` form: a path string, a `build-path` form, or a `lib` form. The *`type-expr`* should produce `'unknown`, `'unknown/mask`, etc., and the default is `'unknown/mask`.

`(include-bitmap/relative-to` *`source file-spec [type-expr]`*`)` SYNTAX

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

8. name-message.ss: Name Message

To load: `(require (lib "name-message.ss" "mrlib"))`

8.1 name-message%

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% (parent _) [(style _)] [(paint-callback _)] [(label _)] [(gl-config _)] [(enabled _)] [(vert-margin _)] [(horiz-margin _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)]) ⇒ name-message% object
  parent : frame%, dialog%, panel%, or pane% object
  style = null : list of symbols in '(border control-border combo vscroll hscroll resize-corner gl deleted no-autoclear transparent)
  paint-callback = void : procedure of two arguments: a canvas% object and a dc<%> object
  label = #f : string (up to 200 characters) or #f
  gl-config = #f : gl-config% object or #f
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  The *style* argument indicates one or more of the following styles:

  - 'border — gives the canvas a thin border
  - 'control-border — gives the canvas a border that is like a text-field% control
  - 'combo — gives the canvas a combo button that is like a combo-field% control; this style is intended for use with 'control-border and not with 'hscroll or 'vscroll
  - 'hscroll — enables horizontal scrolling (initially visible but inactive)
  - 'vscroll — enables vertical scrolling (initially visible but inactive)
  - 'resize-corner — leaves room for a resize control at the canvas's bottom right when only one scrollbar is visible
  - 'gl — *obsolete* (every canvas is an OpenGL context where supported)
  - 'deleted — creates the canvas as initially hidden and without affecting *parent*'s geometry; the canvas can be made active later by calling *parent*'s add-child method
  - 'no-autoclear — prevents automatic erasing of the canvas before calls to on-paint

– `'transparent` — the canvas is automatically “erased” before an update using it’s parent window’s background; the result is undefined if this flag is combined with `'no-autoclear`

The `'hscroll` and `'vscroll` styles create a canvas with an initially inactive scrollbar. The scrollbars are activated with either `init-manual-scrollbars` or `init-auto-scrollbars`, and they can be hidden and re-shown with `show-scrollbars`.

The *`paint-callback`* argument is called by the default `on-paint` method, using the canvas and the DC returned by `get-dc` as the argument.

The *`label`* argument names the canvas for `get-label`, but it is not displayed with the canvas.

The *`gl-config`* argument determines properties of an OpenGL context for this canvas, as obtained through the canvas’s drawing context. See also `get-dc` and `get-gl-context` in `dc<%>`.

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

`on-choose-directory`

- `(send` *`a-name-message`* `on-choose-directory` *`dir`*`)` ⇒ void
  `dir` : path

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

`on-event`

Called when the canvas receives a mouse event. See also “Mouse and Keyboard Events” (§ in *PLT MrEd: Graphical Toolbox Manual*), noting in particular that certain mouse events can get dropped.

- `(send` *`a-name-message`* `on-event` *`event`*`)` ⇒ void
  `event` : `mouse-event%` object

  Handles the click by popping up a menu or message.

`on-paint`

Called when the canvas is exposed or resized so that the image in the canvas can be repainted.

When `on-paint` is called in response to a system expose event and only a portion of the canvas is newly exposed, any drawing operations performed by `on-paint` are clipped to the newly-exposed region; however, the clipping region as reported by `get-clipping-region` does not change.

- `(send` *`a-name-message`* `on-paint``)` ⇒ void

  Draws the control’s current message.

`set-message`

Sets the label for the control.

- `(send` *`a-name-message`* `set-message` *`file-name?`* *`msg`*`)` ⇒ void
  `file-name?` : boolean
  `msg` : path string

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.

8.2 Name-Message Functions

`(calc-button-min-sizes` *`dc string`*`)` PROCEDURE

Calculates the minimum width and height of a button label (when drawn with `draw-frame-button-label`). Returns two values: the width and height.

`(draw-frame-button-label` *`dc string width height inverted`*`)` PROCEDURE

Draws a button label like the one for the (define ...) and filename buttons in the top-left corner of the DrScheme 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 *`string`* should be the string to display on the button. The *`width`* and *`height`* arguments should be the width and height of the button, and *`inverted?`* should be `#t` when the button is pressed.

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

9. plot.ss: Plot

To load: `(require (lib "plot.ss" "mrlib"))`

The **plot.ss** library provides a simple tool for plotting data values to a device context.

Two structures are provided: `data-set` and `plot-setup`.

A `data-set` value includes the following fields (which should be supplied on order to `make-data-set`):

- `points` : `(listof (is-a?/c point%))` — the data values to plot.
- `connected?` : `any?` — indicates whether the points are connected by a line.
- `pen` : `(is-a?/c pen%)` — the drawing pen for plotting points/lines.
- `min-x` : `number?`, `max-x` : `number?`, `min-y` : `number?`, and `max-y` : `number?` — the to plot the points, in drawing-context coordinates.

A `plot-setup` value includes the following fields (which should be supplied on order to `plot-setup`):

- `axis-label-font` : `(is-a?/c font%)` — the font for drawing axis labels.
- `axis-number-font` : `(is-a?/c font%)` — the font for drawing axis numbers.
- `axis-pen` : `(is-a?/c pen%)` — the pen for drawing the axes.
- `grid?` : `any?` — whether to draw a grid at axis markings.
- `grid-pen` : `(is-a?/c pen%)` the pen for drawing the grid (if any).
- `x-axis-marking` : `(listof number?)` — locations for marks on the x-axis.
- `y-axis-marking` : `(listof number?)` — locations for marks on the y-axis.
- `x-axis-label` : `string?` — the x-axis label.
- `y-axis-label` : `string?` — the y-axis label.

`(plot` *`dc data-set-list plot-setup`*`)` PROCEDURE

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

Index