PLT MrEd: Graphical Toolbox Manual

Matthew Flatt
mflatt@cs.utah.edu

Robert Bruce Findler
robby@cs.rice.edu

Version 103
August 2000

Copyright notice

Send us your Web links

If you use any parts or all of the DrScheme 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@cs.rice.edu*. Evidence of interest helps the DrScheme Project to maintain the necessary intellectual and financial support. We appreciate your help.

Thanks

Thanks to Julian Smart for wxWindows and his help. Thanks also to Brent Benson for `libscheme`, and to Hans Boehm for the conservative garbage collector and his help.

The geometry-management classes were originally developed by Richard Cobbe. Thanks also to Shriram Krishnamurthi, Cormac Flanagan, Matthias Felleisen, Paul Steckler, Gann Bierner, Michael Sperber, Dan Grossman, Stephanie Weirich, Sebastian Good, Johnathan Franklin, Mark Krentel, Corky Cartwright, Michael Ernst, Kennis Koldewyn, Bruce Duba, and many others for feedback and help.

This manual was typest using LaTeX and a patched version of `latex2html`. Some typesetting macros were originally taken from Julian Smart's *Reference Manual for wxWindows 1.60: a portable C++ GUI toolkit.*

Contents

10 Editor Procedures   295

IV   Appendices   300

11 Running MrEd   302

Index   305

1.   Introduction

This manual describes the MrEd GUI toolbox for programmers developing MrEd applications. It assumes familiarity with MzScheme as described in *PLT MzScheme: Language Manual* (particularly the class and interface system) and with basic GUI concepts (such as windows and events).

What is MrEd?

MrEd is a Scheme implementation based on MzScheme (see *PLT MzScheme: Language Manual*). MrEd embeds MzScheme and extends it with a graphical user interface (GUI) toolbox. GUI applications written with MrEd run without modification under Windows, MacOS, and Unix/X.

MrEd is *not* a graphical environment for developing Scheme programs. DrScheme, documented in *PLT DrScheme: Development Environment Manual*, is the development environment for producing MzScheme- and MrEd-based applications.[1]

Toolbox Organization

For documentation purposes, the MrEd toolbox is organized into three parts:

- The **windowing toolbox**, for implementing form-filling GUI programs (such as a database query window) using buttons, menus, text fields, and events. The windowing toolbox is described in §2.

- The **drawing toolbox**, for drawing pictures or implementing dynamic GUI programs (such as a video game) using drawing canvases, pens, and brushes. The drawing toolbox is described in §5.

- The **editor toolbox**, for developing traditional text editors, editors that mix text and graphics, or free-form layout editors (such as a word processor, HTML editor, or icon-based file browser). The editor toolbox is described in §8.

These three parts roughly represent layers of increasing sophistication. Simple GUI programs access only the windowing toolbox directly, more complex programs use both the windowing and drawing toolboxes, and large-scale applications rely on all three toolboxes.[2]

All three parts are immediately available when MrEd is started. MrEd's initial Scheme namespace contains bindings for all of the class, interface, and procedure names defined in this manual. In addition, the initial environment binds `mred@` to a signed unit that exports all of the MrEd bindings (including `mred@`) and binds `mred^` to the unit's signature (but no knowledge about units or signatures is required to understand this manual).

---

[1]DrScheme is itself a MrEd-based application that is developed using DrScheme.

[2]This three-layer view of the toolbox breaks down under close scrutiny, because the windowing, drawing, and editor toolboxes are actually interdependent and intertwined. Nevertheless, the layered separation is a good approximation.

Part I

Windowing Toolbox

2.   Windowing Toolbox Overview

MrEd's windowing toolbox provides the basic building blocks of GUI programs, including frames (top-level windows), modal dialogs, menus, buttons, check boxes, text fields, and radio buttons. The toolbox provides these building blocks via built-in classes, such as the `frame%` class:[1]

```
; Make a frame by instantiating the frame% class
(define frame (make-object frame% "Example"))

; Show the frame by calling its show method
(send frame show #t)
```

The built-in classes provide various mechanisms for handling GUI events. For example, when instantiating the `button%` class, the programmer supplies an event callback procedure to be invoked when the user clicks the button. The following example program creates a frame with a text message and a button; when the user clicks the button, the message changes:

```
; Make a frame by instantiating the frame% class
(define frame (make-object frame% "Example"))

; Make a static text message in the frame
(define msg (make-object message% "No events so far..." frame))

; Make a button in the frame
(make-object button% "Click Me" frame
    ; Callback procedure for a button click
    (lambda (button event) (send msg set-label "Button click")))

; Show the frame by calling its show method
(send frame show #t)
```

Programmers never implement the GUI event loop directly. Instead, the system automatically pulls each event from an internal queue and dispatches the event to an appropriate window. The dispatch invokes the window's callback procedure or calls one of the window's methods. In the above program, the system automatically invokes the button's callback procedure whenever the user clicks Click Me.

If a window receives multiple kinds of events, the events are dispatched to methods of the window's class instead of to a callback procedure. For example, a drawing canvas receives update events, mouse events, keyboard events, and sizing events; to handle them, a programmer must derive a new class from the built-in `canvas%` class and override the event-handling methods. The following expression extends the frame created above with a canvas that handles mouse and keyboard events:

```
; Derive a new canvas (a generic drawing window) class to handle events
```

---

[1]To run the example, type it into DrScheme's top window and click the Execute button. (The current language in DrScheme should be MrEd Debug.) Alternatively, save the program to a file using your favorite text editor, and then load it into MrEd via the Load File menu item.

```
(define my-canvas%
  (class canvas% ; The base class is canvas%
         (frame) ; one extra argument, frame, is provided to make-object
    (override
      ; Method to handle mouse events
      [on-event (lambda (event) (send msg set-label "Canvas mouse"))]
      ; Method to handle keyboard events
      [on-char (lambda (event) (send msg set-label "Canvas keyboard"))])
    ; Call the superclass initialization, providing frame
    (sequence (super-init frame))))

; Make a canvas that handles events in the frame
(make-object my-canvas% frame)
```

(It may be neceesary to enlarge the frame to see the new canvas.) Moving the cursor over the canvas calls
the canvas's `on-event` method with an object representing a motion event. Clicking on the canvas calls
`on-event`. While the canvas has the keyboard focus, typing on the keyboard invokes the canvas's `on-char`
method.

The system dispatches GUI events sequentially; that is, after invoking an event-handling callback or method,
the system waits until the handler returns before dispatching the next event. To illustrate the sequential
nature of events, we extend the frame again, adding a Pause button:

```
(make-object button% "Pause" frame (lambda (button event) (sleep 5)))
```

After the user clicks Pause, the entire frame becomes unresponsive for five seconds; the system cannot
dispatch more events until the call to `sleep` returns. For more information about event dispatching, see
§2.4.

In addition to dispatching events, the GUI classes also handle the graphical layout of windows. Our example
frame demonstrates a simple layout; the frame's elements are lined up top-to-bottom. In general, a pro-
grammer specifies the layout of a window by assigning each GUI element to a parent **container**. A vertical
container, such as a frame, arranges its children in a column, and a horizontal container arranges its children
in a row. A container can be a child of another container; for example, to place two buttons side-by-side in
our frame, we create a horizontal panel for the new buttons:

```
(define panel (make-object horizontal-panel% frame))
(make-object button% "Left" panel
    (lambda (button event) (send msg set-label "Left button click")))
(make-object button% "Right" panel
    (lambda (button event) (send msg set-label "Right button click")))
```

For more information about window layout and containers, see §2.2.

2.1   Core Windowing Classes

The fundamental graphical element in MrEd's windowing toolbox is an **area**. The following classes implement
the different types of areas in the windowing toolbox:

- **Containers** — areas that can contain other areas:

    - `frame%` — a **frame** is a top-level window that the user can move and resize.

- – `dialog%` — a **dialog** is a modal top-level window; when a dialog is shown, other top-level windows are disabled until the dialog is dismissed.
- – `panel%` — a **panel** is a subcontainer within a container. The toolbox provides two subclasses of `panel%`: `vertical-panel%` and `horizontal-panel%`.
- – `pane%` — a **pane** is a lightweight panel. It has no graphical representation or event-handling capabilities. The `pane%` class has three subclasses: `vertical-pane%`, `horizontal-pane%`, and `grow-box-spacer-pane%`.

- **Containees** — areas that must be contained within other areas:
  - – `panel%` — a panel is a containee as well as a container.
  - – `pane%` — a pane is a containee as well as a container.
  - – `canvas%` — a **canvas** is a subwindow for drawing on the screen.
  - – `editor-canvas%` — an **editor canvas** is a subwindow for displaying a text editor or pasteboard editor. The `editor-canvas%` class is documented with the editor classes in §8.
  - – **Controls** — containees that the user can manipulate:
    - ∗ `message%` — a **message** is a static text field or bitmap with no user interaction.
    - ∗ `button%` — a **button** is a clickable control.
    - ∗ `check-box%` — a **check box** is a clickable control; the user clicks the control to set or remove its check mark.
    - ∗ `radio-box%` — a **radio box** is a collection of mutually exclusive **radio buttons**; when the user clicks a radio button, it is selected and the radio box's previously selected radio button is deselected.
    - ∗ `choice%` — a **choice item** is a pop-up menu of text choices; the user selects one item in the control.
    - ∗ `list-box%` — a **list box** is a scrollable lists of text choices; the user selects one or more items in the list (depending on the style of the list box).
    - ∗ `text-field%` — a **text field** is a box for simple text entry.
    - ∗ `slider%` — a **slider** is a dragable control that selects an integer value within a fixed range.
    - ∗ `gauge%` — a **gauge** is a output-only control (the user cannot change the value) for reporting an integer value within a fixed range.

As suggested by the above listing, certain areas, called **containers**, manage certain other areas, called **containees**. Some areas, such as panels, are both containers and containees.

Most areas are **windows**, but some are **non-windows**. A window, such as a panel, has a graphical representation,[2] receives keyboard and mouse events, and can be disabled or hidden. In contrast, a non-window, such as a pane, is useful only for geomerty management; a non-window does not receive mouse events, and it cannot be disabled or hidden.

Every area is an instance of the `area<%>` interface. Each container is also an instance of the `area-container<%>` interface, whereas each containee is an instance of `subarea<%>`. Windows are instances of `window<%>`. The `area-container<%>`, `subarea<%>`, and `window<%>` interfaces are subinterfaces of `area<%>`. Figure 2.1 shows more of the type hierarchy under `area<%>`.

Figure 2.2 extends the previous figure to show the complete type hierarchy under `area<%>`.[3] To avoid intersecting lines, the hierarchy is drawn for a cylindrical surface; lines from `subarea<%>` and `subwindow<%>` wrap from the left edge of the diagram to the right edge.

Menu bars, menus, and menu items are graphical elements, but not areas (i.e., they do not have all of the properties that are common to areas, such as an adjustable graphical size). Instead, the menu classes form

---

[2]For a panel, the graphical representation is merely an optional border.

[3]Some of the types in Figure 2.2 are represented by interfaces, and some types are represented by classes. In principle, every area type should be represented by an interface, but whenever the windowing toolbox provides a concrete implementation, the corresponding interface is omitted from the toolbox.

```
                                     area<%>
              ----------------------|---------------
              |                      |               |
          subarea<%>              window<%>      area-container<%>
              |----            -------|----------        |
                  |           |       |          |       |
              subwindow<%>                area-container-window<%>
              --------|--------                       |
              |               |                       |
          control<%>       canvas<%>    top-level-window<%>
```
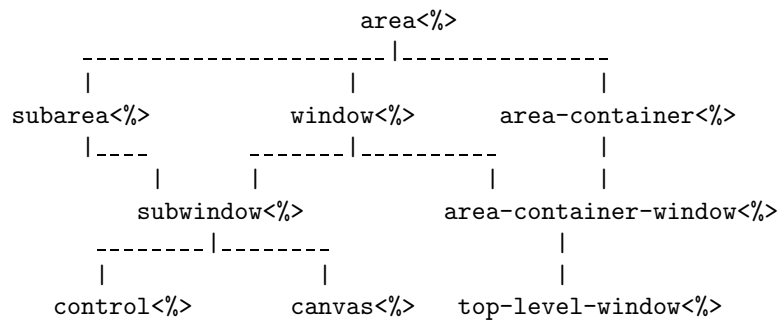
Figure 2.1: Core area type hierarchy

```
                                   area<%>
            ---------------------|---------------
            |                     |               |
        subarea<%>            window<%>        area-container<%>
    <<<----|----          -----|----------     --|---  -------------------<<<
            |           |       |          |     |   | |
        subwindow<%>                       |     |   | |
    <<<-------------|----------        |    |   | |              _<<<
            |                 |    |    |   pane%             |
        control<%>            |    |    |   |- horizontal-pane% |
        |- message%           |    |    |   |- vertical-pane%   |
        |- button%            |    |    |                       |
        |- check-box%         |  area-container-window<%>       |
        |- slider%            |    |                            |
        |- gauge%             |    |        ------------------|
        |- text-field%        |    |        |
        |- radio-box%         |    |-------- panel%
        |- list-control<%>    |    |         |- horizontal-panel%
            |- choice%        |    |         |- vertical-panel%
            |- list-box%      |    |
                              |    |- top-level-window<%>
                              |        |- frame%
                              |        |- dialog%
                          canvas<%>
                          |- canvas%
                          |- editor-canvas%
```
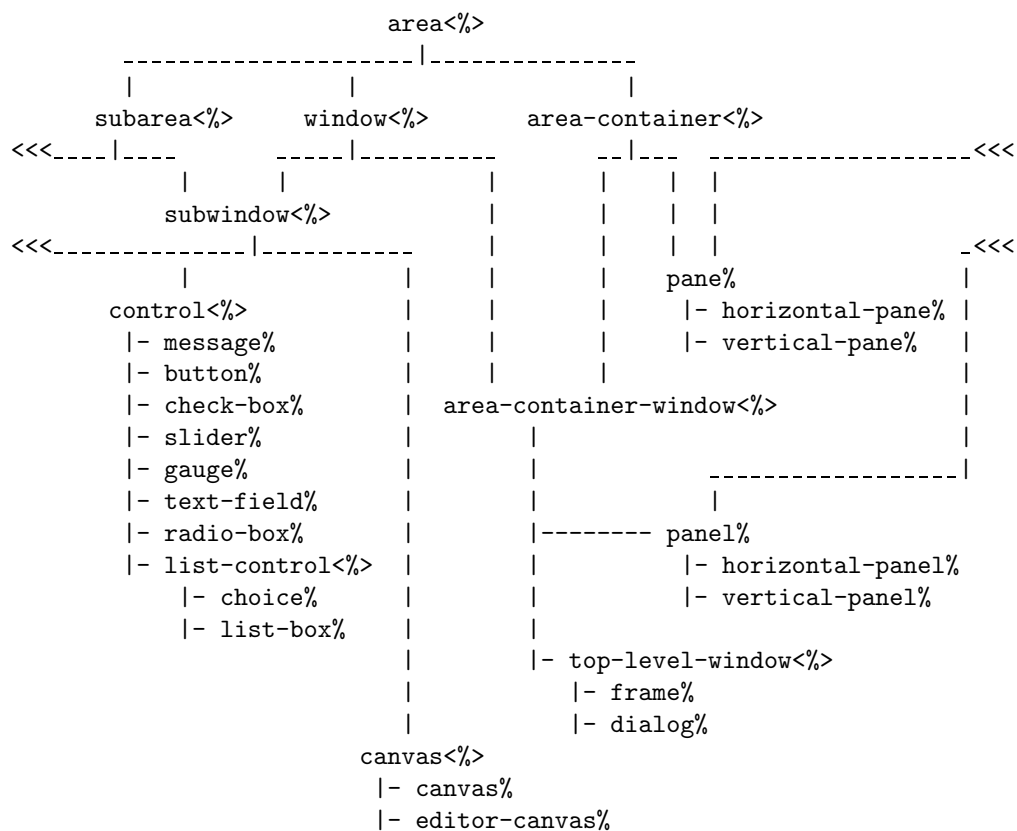
Figure 2.2: Complete area type hierarchy (drawn on a cylinder with wraparound lines)

```
          menu-item<%>                  menu-item-container<%>
               |                                |
               |- separator-menu-item%   _____|___
               |- labelled-menu-item<%>  |        |- menu-bar%
                  ---------|---------     |        |- popup-menu%
                  |                 |     |
                  |                menu%
                  |
               |- selectable-menu-item<%>
                    |- menu-item%
                    |- checkable-menu-item%
```
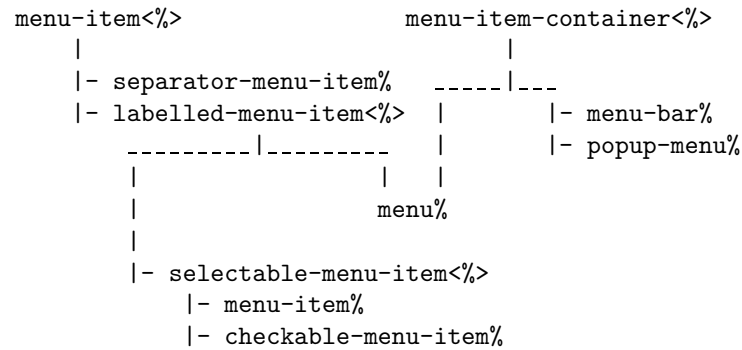
Figure 2.3: Menu type hierarchy

a separate container–containee hierarchy:

- **Menu Item Containers**
  - `menu-bar%` — a **menu bar** is a top-level collection of menus that are associated with a frame.
  - `menu%` — a **menu** contains a set of menu items. The menu can appear in a menu bar, in a popup menu, or as a submenu in another menu.
  - `popup-menu%` — a **popup menu** is a top-level menu that is dynamically displayed in a canvas or editor canvas.

- **Menu Items**
  - `separator-menu-item%` — a **separator** is an unselectable line in a menu or popup menu.
  - `menu-item%` — a **menu item** is a selectable text item in a menu. When the item is selected, its callback procedure is invoked.
  - `checkable-menu-item%` — a **checkable menu item** is a text item in a menu; the user selects a checkable menu item to toggle a check mark next to the item.
  - `menu%` — a menu is a menu item as well as a menu item container.

The complete type hierarchy for the menu system is shown in Figure 2.3.

2.2 Geometry Management

MrEd's geometry management makes it easy to design windows that look right on all platforms, despite different graphical representations of GUI elements. Geometry management is based on containers; each container arranges its children based on simple constraints, such as the current size of a frame and the natural size of a button.

The built-in container classes include horizontal panels (and panes), which align their children in a row, and vertical panels (and panes), which align their children in a column. By nesting horizontal and vertical containers, a programmer can achieve most any layout. For example, we can construct a dialog with the following shape:

```
Your name: [                                    ]

              [Cancel]   [Ok]
```
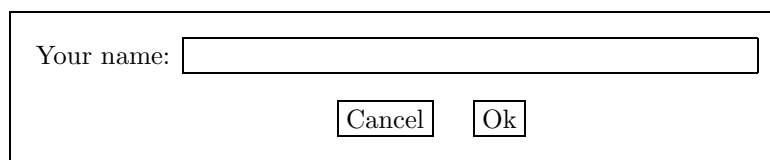
with the following program:

```
; Create a dialog
(define dialog (make-object dialog% "Example"))

; Add a text field to the dialog (with a dummy callback procedure)
(make-object text-field% "Your name" dialog void)
; Note: MzScheme's void procedure accepts any number of arguments

; Add a horizontal panel to the dialog
(define panel (make-object horizontal-panel% dialog))

; Add Cancel and Ok buttons to the horizontal panel
(make-object button% "Cancel" panel void)
(make-object button% "Ok" panel void)

; Change the panel's alignment to center the buttons
(send panel set-alignment 'center 'center)

; Show the dialog
(send dialog show #t)
```

Each container arranges its children using the natural size of each child, which usually depends on instantiation parameters of the child, such as the label on a button or the number of choices in a radio box. In the above example, the dialog stretches horizontally to match the minimum width of the text field, and it strteches vertically to match the total height of the field and the buttons. The dialog then stretches the horizontal panel to fill the bottom half of the dialog. Finally, the horizontal panel uses the sum of the buttons' minimum widths to center them horizontally.

As the example demonstrates, a stretchable container grows to fill its environment, and it distributes extra space among its strechable children. By default, panels are stretchable in both directions, whereas buttons are not stretchable in either direction. The programmer can change whether an individual GUI element is stretchable.

The following subsections describe the container system in detail, first discussing the attributes of a containee in §2.2.1, and then describing the attributes of a container in §2.2.2. In addition to the built-in vertical and horizontal containers, programmers can define new types of containers as discussed in the final subsection, §2.2.3.

2.2.1   Containees

Each containee, or child, has the following properties:

- a graphical minimum width and a graphical minimum height;

- a requested minimum width and a requested minimum height;

- horizontal and vertical stretchability (on or off); and

- horizontal and vertical margins.

A container arranges its chidlren based on these four properties of each containee. A containee's parent container is specified when the containee is created, and the parent cannot be changed. However, a containee can be hidden or inactive within its parent, as described in §2.2.2.

The **graphical minimum size** of a particular containee depends on the platform, the label of the containee (for a control), and style attributes specified when creating the containee. For example, a button's minimum graphical size ensures that the entire text of the label is visible. The graphical minimum size of a control (such as a button) cannot be changed; it is fixed at creation time.[4] The graphical minimum size of a panel or pane depends on the total minimum size of its children and the way that they are arranged.

To select a size for a containee, its parent container considers the containee's **requested minimum size** rather than its graphical minimum size (assuming the requested minimum is larger than the graphical minimum). Unlike the graphical minimum, the requested minimum size of a containee can be changed by a programmer at any time using the `min-width` and `min-height` methods.

Unless a containee is stretchable (in a particular direction), it always shrinks to its minimum size (in the corresponding direction). Otherwise, containees are streched to fill all available space in a container. Each containee begins with a default stretchability. For example, buttons are not initially stretchable, whereas a one-line text field is initially stretchable in the horizontal direction. A programmer can change the stretchability of a containee at any time using the `stretchable-width` and `stretchable-height` methods.

A margin is whitespace surrounding a containee. Each containee's margin is independent of its minimum size, but from the container's point of view, a margin effectively increases the minimum size of the containee. For example, if a button has a vertical margin of 2, then the container must allocate enough room to leave two pixels of whitespace above and below the button, in addition to the space that is allocated for the button's minimum height. A programmer can adjust a containee's margin with `horiz-margin` and `vert-margin`. The default margin is 2 for a control, and 0 for any other type of containee.

In practice, the requested minimum size and margin of a control are rarely changed, although they are often changed for a canvas. Stretchability is commonly adjusted for any type of containee, depending on the visual effect desired by the programmer.

2.2.2   Containers

A container has the following properties:

- a list of (active) children containees;

- a requested minimum width and a requested minimum height;

- a spacing used between the children;

- a border margin used around the total set of children;

- horizontal and vertical stretchability (on or off); and

- an alignment setting for positioning leftover whitespace.

These properties are factored into the container's calculation of its own size and the arrangement of its children. For a container that is also a containee (e.g., a panel), the container's requested minimum size and stretchability are the same as for its containee aspect.

A containee's parent container is specified when the containee is created, and the parent cannot be changed. However, a containee window can be **hidden** or **inactive** within its parent container:[5]

- A hidden child is invisible to the user, but space is still allocated for each hidden child within a container. To hide or show a child, call the child's `show` method.

---

[4]A control's minimum size is *not* recalculated when its label is changed.

[5]A non-window containee cannot be make hidden or inactive.

- An inactive child is hidden *and* ignored by container as it arranges its other children, so no space is reserved in the container for an inactive child. To make a child active or inactive, call the container's `add-child` or `delete-child` method (which calls the child's `show` method).

When a child is created, it is initially shown and active. An inactive child is subject to garbage collection when no external reference to the child exists. A list of active children (hidden or not) is available from a container through its `get-children` method.

The order of the children in a container's active list is significant. For example, a vertical panel puts the first child in its list at the top of the panel, and so on. When a new child is created, it is put at the end of its container's list of children. The order of a container's list can be changed dynamically via the `change-children` method. (The `change-children` method can also be used to activate or deactive children.)

The (graphical) minimum size of a container is calculated by combining the minimum sizes of its children (summing them or taking the maximum, as appropriate to the layout strategy of the container) along with the spacing and border margins of the container. A larger miniumum may be specified by the programmer using `min-width` and `min-height` methods; when the computed minimum for a container is larger than the programmer-specified minimum, then the programmer-specified minimum is ignored.

A container's spacing determines the amount of whitespace left between adjacent children in the container, in addition to any whitespace required by the children's margins. A container's border margin determines the amount of whitespace to add around the collection of children; it effectively decreases the area within the container where children can be placed. A programmer can adjust a container's border and spacing dynamically via the `border` and `spacing` methods. The default border and spacing are `0` for all container types.

Because a panel or pane is a containee as well as a container, it has a containee margin in addition to its border margin. For a panel, these margins are not redundant because the panel can have a graphical border; the border is drawn inside the panel's containee margin, but outside the panel's border margin.

For a top-level-window container, such as a frame or dialog, the container's stretchability determines whether the user can resize the window to something larger than its minimum size. Thus, the user cannot resize a frame that is not stretchable. For other types of containers (i.e., panels and panes), the container's stretachability is its stretchability as a containee in some other container. All types of containers are initially stretchable in both directions,[6] but a programmer can change the stretchability of an area at any time via the `stretchable-width` and `stretchable-height` methods.

The alignment specification for a container determines how it positions its children when the container has leftover space. (A container can only have leftover space in a particular direction when none of its children are stretchable in that direction.) For example, when the container's horizontal alignment is `'left`, the children are left-aligned in the container and leftover whitespace is accumulated to the right. When the container's horizontal alignment is `'center`, each child is horizontally centered in the container. A container's alignment is changed with the `set-alignment` method.

2.2.3 Defining New Types of Containers

Although nested horizontal and vertical containers can express most layout patterns, a programmer can define a new type of container with an explicit layout procedure. A programmer defines a new type of container by deriving a class from `panel%` or `pane%` and overriding the `container-size` and `place-children` methods. The `container-size` method takes a list of size specifications for each child and returns two values: the minimum height and width of the container. The `place-children` method takes the container's size and a

---

[6]Except instances of `grow-box-spacer-pane%`, which is intended as a lightweight spacer class rather than a useful container class.

list of size specifications for each child, and returns a list of sizes and placements (in parallel to the original list).

A input size specification is a list of four values:

- the child's minimum width;

- the child's minimum height;

- the child's horizontal stretchability (`#t` means stretchable, `#f` means not stretchable); and

- the child's vertical stretchability.

For `place-children`, an output position and size specification is a list of four values:

- the child's new horizontal position (relative to the parent);

- the child's new vertical position;

- the child's new actual width;

- the child's new actual height.

The widths and heights for both the input and output include the children's margins. The returned position for each child is automatically incremented to account for the child's margin in placing the control.

2.3   Mouse and Keyboard Events

Whenever the user moves the mouse, clicks or releases a mouse button, or presses a key on the keyboard, an event is generated for some window. The window that receives the event depends on the current state of the graphic display:

- The receiving window of a mouse event is usually the window under the cursor when the mouse is moved or clicked. If the mouse is over a child window, the child window receives the event rather than its parent.

  When the user clicks in a window, the window "grabs" the mouse, so that *all* mouse events go to that window until the mouse button is released (regardless of the location of the cursor). As a result, a user can click on a scrollbar thumb and drag it without keeping the cursor strictly inside the scrollbar control.

- The receiving window of a keyboard event is the window that owns the **keyboard focus** at the time of the event. Only one window owns the focus at any time, and focus ownership is typically displayed by a window in some manner. For example, a text field control shows focus ownership by displaying a blinking caret.

  Within a top-level window, only certain kinds of subwindows can have the focus, depending on the conventions of the platform. Furthermore, the subwindow that initially owns the focus is platform-specific. A user can moves the focus in various ways, usually by clicking the target window. A program can use the `focus` method to move the focus to a subwindow or to set the initial focus.

Controls, such as buttons and list boxes, handle keyboard and mouse events automatically, eventually invoking the callback procedure that was provided when the control was created. A canvas propagates mouse and keyboard events to its `on-event` and `on-char` methods, respectively.

A mouse and keyboard event is delivered in a special way to its window. Each ancestor of the receiving window gets a chance to intercept the event through the `on-subwindow-event` and `on-subwindow-char` methods. See the method descriptions for more information.

The default `on-subwindow-char` method for a top-level window intercepts keyboard events to detect menu-shortcut events and focus-navigation events. See `on-subwindow-char` in `frame%` and `on-subwindow-char` in `dialog%` for details. Certain OS-specific key combinations are captured at a low level, and cannot be overridden. For example, under Windows and X, pressing and releasing Alt always moves the keyboard focus to the menu bar. Similarly, Alt-Tab switches to a different application under Windows.[7]

2.4   Event Dispatching and Eventspaces

A graphical user interface is an inherently multi-threaded system: one thread is the program managing windows on the screen, and the other thread is the user moving the mouse and typing at the keyboard. GUI programs typically use an **event queue** to translate this multi-threaded system into a sequential one, at least from the programmer's point of view. Each user action is handled one at a time, ignoring further user actions until the previous one is completely handled. The conversion from a multi-threaded process to a single-threaded one greatly simplies the implementation of GUI programs.

Despite the programming convenience provided by a purely sequential event queue, certain situations require a less rigid dialog with the user:

- **Nested event handling:** In the process of handling an event, it may be necessary to obtain further information from the user. Usually, such information is obtained via a modal dialog; in whatever fashion the input is obtained, more user events must be received and handled before the original event is completely handled. To allow the further processing of events, the handler for the original event must explicitly **yield** to the system. Yielding causes events to be handled in a nested manner, rather than in a purely sequential manner.

- **Asynchronous event handling:** An application may consist of windows that represent independent dialogs with the user. For example, a drawing program might support multiple drawing windows, and a particularly time-consuming task in one window (e.g., a special filter effect on an image) should not prevent the user from working in a different window. Such an application needs sequential event handling for each individual window, but asynchronous (potentially parallel) event handling across windows. In other words, the application needs a separate event queue for each window, and a separate event-handling thread for each event queue.

In MrEd, an **eventspace** is a context for processing GUI events. Each eventspace maintains its own queue of events, and events in a single eventspace are dispatched sequentially by a designated **handler thread**. An event-handling procedure running in this handler thread can yield to the system by calling `yield`, in which case other event-handling procedures may be called in a nested (but single-threaded) manner within the same handler thread. Events from different eventspaces are dispatched asynchronously by separate handler threads.

When a frame or dialog is created without a parent, it is associated with the *current eventspace* as described in §2.4.3. Events for a top-level window and its decendents are always dispatched in the window's eventspace. Every dialog is modal; a dialog's `show` method implcitly calls `yield` to handle events while the dialog is shown. (See also §2.4.2 for information about threads and modal dialogs.) Furthermore, when a modal dialog is shown, the system disables all other top-level windows in the dialog's eventspace,[8] but windows in

---

[7]Alt-Space invokes the system menu under Windows, but this shortcut is implemented by `on-system-menu-char`, which is called by `on-subwindow-char` in `frame%` and `on-subwindow-char` in `dialog%`.

[8]Disabling a window prevents mouse and keyboard events from reaching the window, but other kinds of events, such as update events, are still delivered.

other eventspaces are unaffected by the modal dialog.

2.4.1   Event Types and Priorities

In addition to events corresponding to user and windowing actions, such as button clicks, key presses, and updates, the system dispatches two kinds of internal events: **timer events** and **explicitly queued events**.

Timer events are created by instances of `timer%`. When a timer is started and then expires, the timer queues an event to call the timer's `notify` method. Like a top-level window, each timer is associated with a particular eventspace (the *current eventspace* as described in §2.4.3) when it is created, and the timer queues the event in its eventspace.

Explicitly queued events are created with `queue-callback`, which accepts a callback procedure to handle the event. The event is enqueued in the current eventspace at the time of the call to `queue-callback`, with either a high or low priority as specified by the (optional) second argument to `queue-callback`.

An eventspace's event queue is actually a priority queue with events sorted according to their kind, from highest-priority (dispatched first) to lowest-priority (dispatched last):

- The highest-priority events are high-priority events installed with `queue-callback`.

- Timer events have the second-highest priority.

- Graphical events, such as mouse clicks or window updates, have the second-lowest priority.

- The lowest-priority events are low-priority events installed with `queue-callback`.

Although a programmer has no direct control over the order in which events are dispatched, a programmer can control the timing of dispatches by setting the event dispatch handler via the `event-dispatch-handler` parameter. This parameter and other eventspace procedures are described in more detail in §4.2.

2.4.2   Eventspaces and Threads

When a new eventspace is created, a corresponding **handler thread** is created for the eventspace. When the system dispatches an event for an eventspace, it always does so in the eventspace's handler thread. A handler procedure can create new threads that run indefinitely, but as long as the handler thread is running a handler procedure, no new events can be dispatched for the corresponding eventspace.

When a handler thread shows a dialog, the dialog's `show` method implicitly calls `yield` for as long as the dialog is shown. When a non-handler thread shows a dialog, the non-handler thread simply blocks until the dialog is dismissed. Calling `yield` with no arguments from a non-handler thread has no effect. Calling `yield` with a semaphore from a non-handler thread is equivalent to calling MzScheme's `semaphore-wait`.

2.4.3   Creating and Setting the Eventspace

Whenever a frame, dialog, or timer is created, it is associated with the eventspace specified by the `current-eventspace` parameter (see parameters, §9.4 in *PLT MzScheme: Language Manual*). When the `current-eventspace` procedure is called with no arguments, it returns the current eventspace value. When `current-eventspace` is called with an eventspace value, it changes the current eventspace to the provided one.

The `make-eventspace` procedure creates a new eventspace. The following example creates a new eventspace and a new frame in the eventspace (the `parameterize` syntactic form temporary sets a parameter value):

```
(let ([new-es (make-eventspace)])
  (parameterize ([current-eventspace new-es])
    (make-object frame% "Example")))
```

When an eventspace is created, it is placed under the management of the current custodian (see parameters, §9.4 in *PLT MzScheme: Language Manual*). When a custodian shuts down an eventspace, all frames and dialogs associated with the eventspace are destroyed (without calling `can-close?` or `on-close` in `top-level-window<%>`), all timers in the eventspace are stopped, and all enqueued callbacks are removed. Attempting to create a new window, timer, or explicitly queued event in a shut-down eventspace raises the `exn:misc` exception.

2.4.4 Exceptions and Continuation Jumps

Whenever the system dispatches an event, the call to the handler procedure is wrapped so that full continuation jumps are not allowed to escape from the dispatch, and escape continuation jumps are blocked at the dispatch site. The following `block` procedure illustrates how the system blocks escape continuation jumps:

```
(define (block f)
  ; calls f, returning (void) if f tries to escape
  (let ([done? #f])
    (let/ec k
      (dynamic-wind
       void
       (lambda () (begin0 (f) (set!  done? #t)))
       (lambda () (unless done? (k (void))))))))

(block (lambda () 5)) ; => 5
(let/ec k (block (lambda () (k 10)))) ; => void
(let/ec k ((lambda () (k 10))) 11) ; => 10
(let/ec k (block (lambda () (k 10))) 11) ; => 11
```

Calls to the event dispatch handler are also protected with `block`.

This blocking of continuation jumps complicates the interaction between `with-handlers` and `yield` (or the default event dispatch handler). For example, in evaluating the expression

```
(with-handlers ([(lambda (x) #t)
                  (lambda (x) (error "error during yield"))])
  (yield))
```

the `"error during yield"` handler is *never* called, even if a callback procedure invoked by `yield` raises an exception. The `with-handlers` expression installs an exception handler that tries to jump back to the context of the `with-handlers` expression before invoking a handler procedure; this jump is blocked by the dispatch within `yield`, so `"error during yield"` is never printed. Exceptions during `yield` are "handled" in the sense that control jumps out of the event handler, but `yield` may dispatch another event rather than escaping or returning.

The following expression demonstrates a more useful way to handle exceptions within `yield`:

```
(let/ec k
  (parameterize ([current-exception-handler
                  (lambda (x)
                    (error "error during yield")
```

```
                    (k))])
    (yield)))
```

This expression installs an exception handler that prints an error message *before* trying to escape. Like the continuation escape associated with `with-handlers`, the escape to `k` never succeeds. Nevertheless, if an exception is raised by an event handler during the call to `yield`, an error message is printed before control returns to the event dispatcher within `yield`.

3.   Windowing Class Reference

3.1   Class Listing

Windows

```
                               area<%>
            --------------------|---------------
            |                   |               |
       subarea<%>          window<%>        area-container<%>
<<<____|____         _____|_____       __|___  ------------------<<<
       |            |                |      |    | | |
       subwindow<%>                  |      |    | | |
<<<_____|_____        |      |    | |          _<<<
       |                      |       |      |    | |           |
    control<%>                |       |      |  pane%           |
     |- message%              |       |      |   |- horizontal-pane% |
     |- button%               |       |      |   |- vertical-pane%   |
     |- check-box%            |  area-container-window<%>            |
     |- slider%               |       |                              |
     |- gauge%                |       |         ------------------|
     |- text-field%           |       |                 |
     |- radio-box%            |       |-------- panel%
     |- list-control<%>  |       |             |- horizontal-panel%
         |- choice%      |       |             |- vertical-panel%
         |- list-box%    |       |
                         |       |- top-level-window<%>
                         |            |- frame%
                         |            |- dialog%
                 canvas<%>
                   |- canvas%
                   |- editor-canvas%
```

Menus

```
menu-item<%>                        menu-item-container<%>
    |                                    |
    |- separator-menu-item%     _____|___
    |- labelled-menu-item<%>  |         |- menu-bar%
        ---------|---------    |         |- popup-menu%
        |                 |    |
        |               menu%
        |
        |- selectable-menu-item<%>
            |- menu-item%
            |- checkable-menu-item%
```

Events

```
control-event%
 |- key-event%
 |- mouse-event%
 |- event%
```

Miscellaneous

```
clipboard<%>
clipboard-client%
cursor%
timer%
```

3.2   `area<%>`

An `area<%>` object is either a window or a windowless container for managing the position and size of other areas. An `area<%>` can be a container, a containee, or both. The only areas without a parent are top-level windows.

`get-graphical-min-size`

Returns the area's graphical minimum size as two values: the minimum width and the minimum height (in pixels).

See §2.2 for more information. Note that the return value *does not* depend on the area's `min-width` and `min-width` settings.

  - (send *an-area* get-graphical-min-size) ⇒ two exact integers in [0, 10000]

`get-parent`

Returns the area's parent. A top-level window may have no parent (in which case `#f` is returned), or it may have another top-level window as its parent.

  - (send *an-area* get-parent) ⇒ area<%> object or #f

`get-top-level-window`

Returns the area's closest frame or dialog ancestor. For a frame or dialog area, the frame or dialog itself is returned.

- (send *an-area* `get-top-level-window`) ⇒ `frame%` or `dialog%` object

`min-height`

Gets or sets the area's minimum height for geometry management.

The minimum height is ignored when it is smaller than the area's *minimum graphical height*, or when it is smaller than the height reported by `container-size` if the area is a container. See §2.2 for more information.

An area's initial minimum height is its graphical minimum height. See also `get-graphical-min-size` .

- (send *an-area* `min-height`) ⇒ exact integer in [0, 10000]

  Returns the current minimum height (in pixels).

- (send *an-area* `min-height` *h*) ⇒ void
    h : exact integer in [0, 10000]

  Sets the minimum height (in pixels); if *h* is smaller than the internal hard minimum, an `exn:application:mismatch` exception is raised.

`min-width`

Gets or sets the area's minimum width (in pixels) for geometry management.

The minimum width is ignored when it is smaller than the area's *minimum graphical width*, or when it is smaller than the width reported by `container-size` if the area is a container. See §2.2 for more information.

An area's initial minimum width is its graphical minimum width. See also `get-graphical-min-size` .

- (send *an-area* `min-width`) ⇒ exact integer in [0, 10000]

  Returns the current minimum width (in pixels).

- (send *an-area* `min-width` *w*) ⇒ void
    w : exact integer in [0, 10000]

  Sets the minimum width (in pixels); if *w* is smaller than the internal hard minimum, an `exn:application:mismatch` exception is raised.

`stretchable-height`

Gets or sets the area's vertical stretchability for geometry management. See §2.2 for more information.

- (send *an-area* `stretchable-height`) ⇒ boolean

  Returns the current vertical stretchability.

- (send *an-area* `stretchable-height` *stretch?*) ⇒ void
    stretch? : boolean

  Sets the vertical stretchability.

`stretchable-width`

Gets or sets the area's horizontal stretchability for geometry management. See §2.2 for more information.

- (send *an-area* `stretchable-width`) $\Rightarrow$ boolean

  Returns the current horizontal stretchability.

- (send *an-area* `stretchable-width` *stretch?*) $\Rightarrow$ void
     stretch? : boolean

  Sets the horizontal stretchability.

3.3   `area-container<%>`

Extends: `area<%>`

A `subarea<%>` is a container `area<%>`.

`add-child`

Add the given subwindow to the set of active children. See also `change-children`.

- (send *an-area-container* `add-child` *child*) $\Rightarrow$ void
     child : subwindow<%> object

`after-new-child`

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

- (send *an-area-container* `after-new-child` *child*) $\Rightarrow$ void
     child : subarea<%> object

  Does nothing.

`begin-container-sequence`

Suspends geometry management in the container's top-level window until `end-container-sequence` is called. The `begin-container-sequence` and `end-container-sequence` methods are used to bracket a set of container modifications so that the resulting geometry is computed only once. The commands may be nested arbitrarily deep.

- (send *an-area-container* `begin-container-sequence`) $\Rightarrow$ void

`border`

Gets or sets the border margin for the container in pixels. This margin is used as an inset into the panel's client area before the locations and sizes of the subareas are computed.

- (send *an-area-container* `border`) ⇒ exact integer in [0, 1000]

  Returns the current border margin.

- (send *an-area-container* `border` *margin*) ⇒ void
    margin : exact integer in [0, 1000]

  Sets the border margin.

change-children

Takes a filter procedure and changes the container's list of active children. The filter procedure takes a list of children areas and returns a new list of children areas. The new list must consist of children that were created as subareas of this area (i.e., `change-children` cannot be used to change the parent of a subarea).

After the set of active children is changed, the container computes the sets of newly inactive and newly active children. Newly inactive windows are hidden. Newly active windows are shown.

Since non-window areas cannot be hidden, non-window areas cannot be made inactive. If the filter procedure removes non-window subareas, an exception is raised and the set of active children is not changed.

- (send *an-area-container* `change-children` *filter*) ⇒ void
    filter : procedure of one argument, a list of `subarea<%>` objects, that returns a list of
             `subarea<%>` objects

container-size

Called to determine the minimum size of a container. See §2.2 for more information.

- (send *an-area-container* `container-size` *info*) ⇒ two exact integers in [0, 10000]
    info : list of list containing two exact integers in [0, 10000] and two booleans

delete-child

Removes the given subwindow from the list of active children. See also `change-children` .

- (send *an-area-container* `delete-child` *child*) ⇒ void
    child : `subwindow<%>` object

end-container-sequence

See `begin-container-sequence`.

- (send *an-area-container* `end-container-sequence`) ⇒ void

get-alignment

Returns the container's current alignment specification. See `set-alignment` for more information.

- (send *an-area-container* `get-alignment`) ⇒ two symbols

`get-children`

Returns a list of the container's active children. (The active children are the ones currently managed by the container; inactive children are generally hidden.) The order of the children in the list is significant. For example, in a vertical panel, the first child in the list is placed at the top of the panel.

- (send *an-area-container* `get-children`) ⇒ list of `subarea<%>` objects

`place-children`

Called to place the children of a container. See §2.2 for more information.

- (send *an-area-container* `place-children` *info width height*) ⇒ list of list containing four exact integers in [0, 10000]
    info : list of list containing two exact integers in [0, 10000] and two booleans
    width : exact integer in [0, 10000]
    height : exact integer in [0, 10000]

`reflow-container`

When a container window is not shown, changes to the container's set of children do not necessarily trigger the immediate re-computation of the container's size and its chidlren's positions. Instead, the recalculation is delayed until the container is shown, which avoids redundant computations between a series of changes. The `reflow-container` method forces the immediate recalculation of the container's and its childrens's sizes and locations.

Immediately after calling the `reflow-container` method , `get-width`, `get-height`, `get-x`, and `get-y` report the correct size and location for the container and its children, even when the container is hidden.

- (send *an-area-container* `reflow-container`) ⇒ void

`set-alignment`

Sets the alignment specification for a container, which determines how it positions its children when the container has leftover space (when a child was not stretchable in a particular dimension).

When the container's horizontal alignment is `'left`, the children are left-aligned in the container and whitespace is inserted to the right. When the container's horizontal alignment is `'center`, each child is horizontally centered in the container. When the container's horizontal alignment is `'right`, leftover whitespace is inserted to the left.

Similarly, a container's vertical alignment can be `'top`, `'center`, or `'bottom`.

- (send *an-area-container* `set-alignment` *horiz-align vert-align*) ⇒ void
    horiz-align : symbol in `'(left center right)`
    vert-align : symbol in `'(top center bottom)`

`spacing`

Gets or sets the spacing, in pixels, used between subareas in the container. For example, a vertical panel inserts this spacing between each pair of vertically aligned subaraes (with no extra space at the top or bottom).

- (send *an-area-container* `spacing`) ⇒ exact integer in [0, 1000]

  Returns the current spacing.

- (send *an-area-container* `spacing` *spacing*) ⇒ void
    spacing : exact integer in [0, 1000]

  Sets the spacing.

3.4 `area-container-window<%>`

Extends: `area-container<%>`

Extends: `window<%>`

`get-control-font`

See `set-control-font`.

- (send *an-area-container-window* `get-control-font`) ⇒ font% object

`get-label-font`

See `set-label-font`.

- (send *an-area-container-window* `get-label-font`) ⇒ font% object

`get-label-position`

Returns the current label arrangement for the container. See `set-label-position`.

- (send *an-area-container-window* `get-label-position`) ⇒ symbol in '(horizontal vertical)

`set-control-font`

Sets the font for drawing:

- buttons labels,

- check box labels,

- radio button labels,

- choice and list box items,

- text field contents, and

- slider and gauge values (if any)

for newly-created controls within the container.

Only controls and sub-containers created after the call to `set-control-font` are affected. When a child container window is created, it inherits the label font setting of its parent.

See also `set-label-font`.

- (send *an-area-container-window* `set-control-font` *font*) ⇒ void
    font : `font%` object

set-label-font

Sets the font for drawing

- message labels,
- radio box labels (not the individual buttons),
- choice and list box labels (not the selectable items),
- text field labels (not the contents), and
- slider and gauge labels.

in newly-created controls within the container.

Only controls and sub-containers created after the call to `set-label-font` are affected. When a child container window is created, it inherits the label font setting of its parent.

See also `set-control-font`.

- (send *an-area-container-window* `set-label-font` *return*) ⇒ void
    return : `font%` object

set-label-position

Sets the arrangement of labels, `'horizontal` (to the left of the control) or `'vertical` (above the control), for radio boxes, choice items, list boxes, text fields, sliders and gauges. Button and check box labels are not affected.

Only controls and sub-containers created after the call to `set-label-position` are affected. When a child container window is created, it inherits the label position setting of its parent. Horizontal label placement is the default placement for a top-level window.

- (send *an-area-container-window* `set-label-position` *position*) ⇒ void
    position : symbol in `'(horizontal vertical)`

3.5   `button%`

Implements: `control<%>`

Whenever a button is clicked by the user, the buttons's callback procedure is invoked. A callback procedure is provided as an initialization argument when each button is created.

- (make-object button% *label parent callback style*) ⇒ button% object
    label : string or bitmap% object
    parent : frame%, dialog%, panel%, or pane% object
    callback : procedure of two arguments: a button% object and a control-event% object
    style = null : list of symbols in '(border)

  Creates a button with a string or bitmap label. If *label* is a bitmap, then the bitmap must be valid (see ok? in bitmap%) and not installed in a bitmap-dc% object; otherwise, an exn:application:mismatch exception is raised.

  If an ampersand ("&") occurs in *label* (when *label* is a string), it is specially parsed; under Windows and X, the character following an ampersand is underlined in the displayed control to indicate a keyboard mnemonic. (Under MacOS, mnemonic underlines are not shown.) The underlined mnemonic character must be a letter or a digit. The user can effectively click the button by typing the mnemonic when the control's top-level-window contains the keyboard focus. The user must also hold down the Meta or Alt key if the keyboard focus is currently in a control that handles normal alphanumeric input. The ampersand itself is removed from *label* before it is displayed for the control; a double-ampersand in *label* is converted to a single ampersand (with no mnemonic underlining). Mnemonic keyboard events are handled by on-traverse-char (but not under MacOS).

  The *callback* procedure is called (with the event type 'button) whenever the user clicks the button.

  If *style* includes 'border, the button is drawn with a special border that indicates to the user that it is the default action button (see on-traverse-char).

`set-label`

Sets a window's label. The window's natural minimum size might be different after the label is changed, but the window's mininum size is not recomputed.

See get-label for more information.

- (send *a-button* set-label *label*) ⇒ void
    label : bitmap% object

  Sets the bitmap label for a bitmap button. Since *label* is a bitmap, the bitmap must be valid (see ok? in bitmap%) and not installed in a bitmap-dc% object; otherwise, an exn:application:mismatch exception is raised. The bitmap label is installed only if the control was originally created with a bitmap label.

- (send *a-button* set-label *l*) ⇒ void
    l : string or #f

  If *l* is #f, the window's label is removed.

3.6 canvas<%>

Extends: subwindow<%>

A canvas is a subwindow onto which graphics and text can be drawn. Canvases also receive mouse and keyboard events.

To draw onto a canvas, get its device context (see `get-dc`).

The `canvas<%>` interface is implemented by two classes:

- `canvas%` — a canvas for arbitrary drawing and event handling

- `editor-canvas%` — a canvas for displaying `editor<%>` objects

get-dc

Gets the canvas's device context. See `dc<%>` for more information about drawing.

- (send *a-canvas* `get-dc`) $\Rightarrow$ `dc<%>` object

min-client-height

Gets or sets the canvas's minimum height for geometry management, based on the client size rather than the full size. The client height is obtained or changed via `min-height` in `area<%>`, adding or subtracting border and scrollbar sizes as appropriate.

The minimum height is ignored when it is smaller than the canvas's *minimum graphical height*. See §2.2 for more information.

- (send *a-canvas* `min-client-height`) $\Rightarrow$ exact integer in [0, 10000]

   Returns the current minimum client height (in pixels).

- (send *a-canvas* `min-client-height` *h*) $\Rightarrow$ void
   h : exact integer in [0, 10000]

   Sets the minimum client height (in pixels).

min-client-width

Gets or sets the canvas's minimum width for geometry management, based on the canvas's client size rather than its full size. The client width is obtained or changed via `min-width` in `area<%>`, adding or subtracting border and scrollbar sizes as appropriate.

The minimum width is ignored when it is smaller than the canvas's *minimum graphical width*. See §2.2 for more information.

- (send *a-canvas* `min-client-width`) $\Rightarrow$ exact integer in [0, 10000]

   Returns the current minimum client width (in pixels).

- (send *a-canvas* `min-client-width` *w*) $\Rightarrow$ void
   w : exact integer in [0, 10000]

   Sets the minimum client width (in pixels).

on-char

Called when the canvas receives a keyboard event. See also section 2.3 (page 12).

- (send *a-canvas* `on-char` *ch*) ⇒ void
    ch : `key-event%` object

   Does nothing.

on-event

Called when the canvas receives a mouse event. See also section 2.3 (page 12).

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

   Does nothing.

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-canvas* `on-paint`) ⇒ void

   Does nothing.

on-scroll

Called when the user changes one of the canvas's manual scrollbars. A `scroll-event%` argument provides information about the scroll action.

This method is not called when automatic scrollbars are changed; the `on-paint` method is called instead.

- (send *a-canvas* `on-scroll` *event*) ⇒ void
    event : `scroll-event%` object

on-tab-in

Called when the keyboard focus enters the canvas via keyboard navigation events. The `on-focus` method is also called, as usual for a focus change. When the keyboard focus leaves a canvas due to a navgation event, only `on-focus` is called.

See also `accept-tab-focus` in `canvas%` and `on-traverse-char` in `top-level-window<%>` .

- (send *a-canvas* `on-tab-in`) ⇒ void

   Does nothing.

popup-menu

Pops up the given `popup-menu%` object at the specified coordinates (in this window's coordinates), and returns after handling an unspecified number of events; the menu may still be popped up when this method

returns. If a menu item is selected from the popup-menu, the callback for the menu item is called. (The eventspace for menu item's callback is the canvas's eventspace.)

While the menu is popped up, its target is set to the canvas. See `get-popup-target` for more information.

- (send *a-canvas* `popup-menu` *menu x y*) $\Rightarrow$ void
    menu : `popup-menu%` object
    x : exact integer in [0, 10000]
    y : exact integer in [0, 10000]

  The *menu* is popped up within the window at position (x, y).

`warp-pointer`

Moves the cursor to the given location on the canvas.

- (send *a-canvas* `warp-pointer` *x y*) $\Rightarrow$ void
    x : exact integer in [0, 10000]
    y : exact integer in [0, 10000]

3.7   `canvas%`

Implements: `canvas<%>`

A `canvas%` object is a general-purpose window for drawing and handling events.

- (`make-object` `canvas%` *parent style*) $\Rightarrow$ `canvas%` object
    parent : `frame%`, `dialog%`, `panel%`, or `pane%` object
    style = `null` : list of symbols in '(`border vscroll hscroll`)

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

    − '`border` — gives the canvas a thin border
    − '`hscroll` — enables horizontal scrolling (initially inactive)
    − '`vscroll` — enables vertical scrolling (initially inactive)

  The '`hscroll` and '`vscroll` styles create a canvas with an initially inactive scrollbar. The scrollbar is activated with either `init-manual-scrollbars` or `init-auto-scrollbars`.

`accept-tab-focus`

Gets or sets whether tab-focus is enabled for the canvas. When tab-focus is enabled, the canvas can receive the keyboard focus when the user navigates among a frame or dialog's controls with the Tab and arrow keys. By default, tab-focus is disabled.

When tab-focus is enabled for a canvas, Tab, arrow, and Enter keyboard events are consumed by a frame's default `on-traverse-char` method. (In addition, a dialog's default method consumes Escape key events.) Otherwise, `on-traverse-char` allows the keyboard events to be propagated to the canvas.

- (send *a-canvas* `accept-tab-focus`) $\Rightarrow$ boolean

  Returns `#t` if tab-focus is enabled for the canvas, `#f` otherwise.

- (send *a-canvas* `accept-tab-focus` *on?*) ⇒ void
    on? : boolean

Enables or disables tab-focus for the canvas.

`get-scroll-page`

Get the current page step size of a manual scrollbar. The result is `0` if the scrollbar is not active or it is automatic.

See also `init-manual-scrollbars`.

- (send *a-canvas* `get-scroll-page` *which*) ⇒ exact integer in [1, 10000]
    which : symbol in '(horizontal vertical)

The `which` argument is either '`horizontal` or '`vertical`, indicating whether to get the page step size of the horizontal or vertical scrollbar, repsectively.

`get-scroll-pos`

Gets the current value of a manual scrollbar. The result is always `0` if the scrollbar is not active or it is automatic.

See also `init-manual-scrollbars`.

- (send *a-canvas* `get-scroll-pos` *which*) ⇒ exact integer in [0, 10000]
    which : symbol in '(horizontal vertical)

The `which` argument is either '`horizontal` or '`vertical`, indicating that the value of the horizontal or vertical scrollbar should be returned, repsectively.

`get-scroll-range`

Gets the current maximum value of a manual scrollbar. The result is always `0` if the scrollbar is not active or it is automatic.

See also `init-manual-scrollbars`.

- (send *a-canvas* `get-scroll-range` *which*) ⇒ exact integer in [0, 10000]
    which : symbol in '(horizontal vertical)

The `which` argument is either '`horizontal` or '`vertical`, indicating whether to get the maximum value of the horizontal or vertical scrollbar, repsectively.

`get-view-start`

Get the location at which the visible portion of the canvas starts, based on the current values of the horizontal and vertical scrollbars if they are initialized as automatic (see `init-auto-scrollbars`). Combined with `get-client-size`, an application can efficiently redraw only the visible portion of the canvas. The values are in pixels.

If the scrollbars are disabled or initialized as manual (see `init-manual-scrollbars`), the result is 0.

- (send *a-canvas* `get-view-start`) ⇒ two exact integers in [0, 10000]

`get-virtual-size`

Gets the size in device units of the scrollable canvas area (as opposed to the client size, which is the area of the canvas currently visible). This is the same size as the client size (as returned by `get-client-size`) unless scrollbars are initialized as automatic (see `init-auto-scrollbars`).

  - (send *a-canvas* `get-virtual-size`) ⇒ two exact integers in [0, 10000]

`init-auto-scrollbars`

Enables and initializes automatic scrollbars for the canvas. A horizontal or vertical scrollbar can be activated only in a canvas that was created with the `'hscroll` or `'vscroll` style flag, respectively.

With automatic scrollbars, the programmer specifies the desired virtual size of the canvas, and the scrollbars are automatically handled to allow the user to scroll around the virtual area.

See also `init-manual-scrollbars` for information about manual scrollbars. The horizontal and vertical scrollbars are always either both manual or both automatic, but they are independently enabled. Automatic scrollbars can be re-initialized as manual, and vice-versa.

  - (send *a-canvas* `init-auto-scrollbars` *horiz-pixels vert-pixels h-value v-value*) ⇒ void
     horiz-pixels : exact integer in [1, 10000] or `#f`
     vert-pixels : exact integer in [1, 10000] or `#f`
     h-value : real number in [0.0, 1.0]
     v-value : real number in [0.0, 1.0]

    Initializes the scrollbars and resets the canvas's virtual size to the given values. If either *horiz-pixels* or *vert-pixels* is `#f`, the scrollbar is not enabled in the corresponding direction, and the canvas's virtual size in that direction is the same as its client size.

    The *h-value* and *v-value* arguments specify the initial values of the scrollbars as a fraction of the scrollbar's range. A `0.0` value initializes the scrollbar to its left/top, while a `1.0` value initializes the scrollbar to its right/bottom.

    See also `on-scroll` and `get-virtual-size`.

`init-manual-scrollbars`

Enables and initializes manual scrollbars for the canvas. A horizontal or vertical scrollbar can be activated only in a canvas that was created with the `'hscroll` or `'vscroll` style flag, respectively.

With manual scrollbars, the programmer is responsible for managing all details of the scrollbars, and the scrollbar state has no effect on the canvas's virtual size. Instead, the canvas's virtual size is the same as its client size.

See also `init-auto-scrollbars` for information about automatic scrollbars. The horizontal and vertical scrollbars are always either both manual or both automatic, but they are independently enabled. Automatic scrollbars can be re-initialized as manual, and vice-versa.

  - (send *a-canvas* `init-manual-scrollbars` *h-length v-length h-page v-page h-value v-value*) ⇒ void
     h-length : exact integer in [0, 10000] or `#f`
     v-length : exact integer in [0, 10000] or `#f`
     h-page : exact integer in [1, 10000]
     v-page : exact integer in [1, 10000]

> *h-value* : exact integer in [0, 10000]
> *v-value* : exact integer in [0, 10000]

The *h-length* and *v-length* arguments specify the length of each scrollbar in scroll steps (i.e., the maximum value of each scrollbar). If either is `#f`, the scrollbar is disabled in the corresponding direction.

The *h-page* and *v-page* arguments set the number of scrollbar steps in a page, i.e., the amount moved when pressing above or below the value indicator in the scrollbar control.

The *h-value* and *v-value* arguments specify the initial values of the scrollbars.

If *h-value* is greater than *h-length* or *v-value* is greater than *v-length*, an `exn:application:mismatch` exception is raised. (The page step may be larger than the total size of a scrollbar.)

See also `on-scroll` and `get-virtual-size`.

scroll

Sets the values of automatic scrollbars. (This method has no effect on manual scrollbars.)

> - (send *a-canvas* `scroll` *h-value* *v-value*) ⇒ void
>   *h-value* : real number in [0.0, 1.0] or `#f`
>   *v-value* : real number in [0.0, 1.0] or `#f`

If either argument is `#f`, the scrollbar value is not changed in the corresponding direction.

The *h-value* and *v-value* arguments each specify a fraction of the scrollbar's movement. A `0.0` value sets the scrollbar to its left/top, while a `1.0` value sets the scrollbar to its right/bottom. A `0.5` value sets the scrollbar to its middle. In general, if the canvas's virtual size is *v*, its client size is *c*, and (`>` *v c*), then scrolling to *p* sets the view start to (`floor` (`*` *p* (`-` *v c*))).

See also `init-auto-scrollbars`and `get-view-start`.

set-scroll-page

Set the current page step size of a manual scrollbar. (This method has no effect on automatic scrollbars.)

See also `init-manual-scrollbars`.

> - (send *a-canvas* `set-scroll-page` *which value*) ⇒ void
>   *which* : symbol in '(`horizontal vertical`)
>   *value* : exact integer in [1, 10000]

The `which` argument is either '`horizontal` or '`vertical`, indicating whether to set the page step size of the horizontal or vertical scrollbar, repsectively.

set-scroll-pos

Sets the current value of a manual scrollbar. (This method has no effect on automatic scrollbars.)

The value of the canvas's scrollbar can be changed by the user scrolling, and such changes do not go through this method; use `on-scroll` to monitor scrollbar value changes.

See also `init-manual-scrollbars` and `scroll`.

> - (send *a-canvas* `set-scroll-pos` *which value*) ⇒ void
>   *which* : symbol in '(`horizontal vertical`)
>   *value* : exact integer in [0, 10000]

The `which` argument is either `'horizontal` or `'vertical`, indicating whether to set the value of the horizontal or vertical scrollbar set, repsectively.

set-scroll-range

Sets the current maximum value of a manual scrollbar. (This method has no effect on automatic scrollbars.)

See also `init-manual-scrollbars`.

- (send *a-canvas* `set-scroll-range` *which value*) $\Rightarrow$ void
  which : symbol in `'(horizontal vertical)`
  value : exact integer in [0, 10000]

The `which` argument is either `'horizontal` or `'vertical`, indicating whether to set the maximum value of the horizontal or vertical scrollbar, repsectively.

3.8   `check-box%`

Implements: `control<%>`

A check box is a labelled box which is either checked or unchecked.

Whenever a check box is clicked by the user, the check box's value is toggled and its callback procedure is invoked. A callback procedure is provided as an initialization argument when each check box is created.

- (make-object `check-box%` *label parent callback style*) $\Rightarrow$ `check-box%` object
  label : string or `bitmap%` object
  parent : `frame%`, `dialog%`, `panel%`, or `pane%` object
  callback : procedure of two arguments: a `check-box%` object and a `control-event%` object
  style = `null` : an empty list of symbols

Creates a check box with a string or bitmap label. If *label* is a bitmap, then the bitmap must be valid (see `ok?` in `bitmap%`) and not installed in a `bitmap-dc%` object; otherwise, an `exn:application:mismatch` exception is raised.

If an ampersand ("&") occurs in *label* (when *label* is a string), it is specially parsed; under Windows and X, the character following an ampersand is underlined in the displayed control to indicate a keyboard mnemonic. (Under MacOS, mnemonic underlines are not shown.) The underlined mnemonic character must be a letter or a digit. The user can effectively click the check box by typing the mnemonic when the control's top-level-window contains the keyboard focus. The user must also hold down the Meta or Alt key if the keyboard focus is currently in a control that handles normal alphanumeric input. The ampersand itself is removed from *label* before it is displayed for the control; a double-ampersand in *label* is converted to a single ampersand (with no mnemonic underlining). Mnemonic keyboard events are handled by `on-traverse-char` (but not under MacOS).

The *callback* procedure is called (with the event type `'check`) whenever the user clicks the check box.

The *style* argument is provided for future extensions. Currently, *style* must be the empty list.

get-value

Gets the state of the check box: `#t` if it is checked, `#f` otherwise.

- (send *a-check-box* `get-value`) $\Rightarrow$ boolean

`set-label`

Sets a window's label. The window's natural minimum size might be different after the label is changed, but the window's mininum size is not recomputed.

See `get-label` for more information.

- (send *a-check-box* `set-label` *label*) ⇒ void
    label : `bitmap%` object

  Since *label* is a bitmap, the bitmap must be valid (see `ok?` in `bitmap%`) and not installed in a `bitmap-dc%` object; otherwise, an `exn:application:mismatch` exception is raised. The bitmap label is installed only if the control was originally created with a bitmap label.

- (send *a-check-box* `set-label` *l*) ⇒ void
    l : string or `#f`

  If *l* is `#f`, the window's label is removed.

`set-value`

Sets the check box's state. (The control's callback procedure is *not* invoked.)

The check box's state can be changed by the user clicking the control, and such changes do not go through this method; use the control callback procedure (provided as an initialization argument) to monitor state changes.

- (send *a-check-box* `set-value` *state*) ⇒ void
    state : boolean

  If *state* is `#f`, the box is unchecked, otherwise it is checked.

3.9   `checkable-menu-item%`

Implements: `selectable-menu-item<%>`

A `checkable-menu-item%` is a string-labelled menu item that maintains a checkmark. Its parent must be a `menu%` or `popup-menu%`. When the user selects the menu item, the item's checkmark is toggled and its callback procedure is called.

- (make-object `checkable-menu-item%` *label parent callback shortcut help*) ⇒ `checkable-menu-item%` object
    label : string
    parent : `menu%` or `popup-menu%` object
    callback : procedure of two arguments: a `menu-item%` object and a `control-event%` object
    shortcut = `#f` : character or `#f`
    help = `#f` : string or `#f`

  Creates a new menu item in *parent*. The item is initially shown, appended to the end of its parent, and unchecked. The *callback* procedure is called (with the event type `'menu`) when the menu item is selected (either via a menu bar or `popup-menu` in `canvas<%>`).

  See `set-label` for information about mnemonic ampersands ("&") in *label*.

  If shortcut is not `#f`, the item has a shortcut. See `get-shortcut` for more information.

  If help is not `#f`, the item has a help string. See `get-help-string` for more information.

`check`

Checks or unchecks the menu item.

A menu item's check state can be changed by the user selecting the item, and such changes do not go through this method; use the menu item callback procedure (provided as an initialization argument) to monitor check state changes.

- (send *a-checkable-menu-item* `check` *check?*) ⇒ `void`
    check? : `boolean`

`is-checked?`

Returns `#t` if the item is checked, #f otherwise.

- (send *a-checkable-menu-item* `is-checked?`) ⇒ `boolean`

3.10 `choice%`

Implements: `list-control<%>`

A choice item allows the user to select one string item from a pop-up list of items. Unlike a list box, only the currently selection is visible until the user pops-up the menu of choices.

Whenever the selection of a choice item is changed by the user, the choice item's callback procedure is invoked. A callback procedure is provided as an initialization argument when each choice item is created.

See also `list-box%`.

- (make-object `choice%` *label choices parent callback style*) ⇒ `choice%` object
    label : `string or #f`
    choices : `list of strings`
    parent : `frame%`, `dialog%`, `panel%`, or `pane%` object
    callback : procedure of two arguments: a `choice%` object and a `control-event%` object
    style = `null` : an empty list of symbols
  Creates a choice item. If *label* is a string, it is used as the label for the choice item.

  If an ampersand ("&") occurs in *label*, it is specially parsed; under Windows and X, the character following an ampersand is underlined in the displayed control to indicate a keyboard mnemonic. (Under MacOS, mnemonic underlines are not shown.) The underlined mnemonic character must be a letter or a digit. The user can move the keyboard focus to the choice item by typing the mnemonic when the control's top-level-window contains the keyboard focus. The user must also hold down the Meta or Alt key if the keyboard focus is currently in a control that handles normal alphanumeric input. The ampersand itself is removed from *label* before it is displayed for the control; a double-ampersand in *label* is converted to a single ampersand (with no mnemonic underlining). Mnemonic keyboard events are handled by `on-traverse-char` (but not under MacOS).

  The *choices* list specifies the initial list of user-selectable items for the control. The initial set of choices determines the control's minimum graphical width (see §2.2 for more information).

  The *callback* procedure is called (with the event type `'choice`) when the user selects a choice item (or re-selects the currently selected item).

  The *style* argument is provided for future extensions. Currently, *style* must be the empty list.

3.11 `clipboard<%>`

There is one `clipboard<%>` object, `the-clipboard`, which manages the contents of the system-wide clipboard for cut-and-paste.

Data can be entered into the clipboard in one of two ways: by setting the current clipboard string, or by installing a `clipboard-client%` object. When a client is installed, requests for clipboard data are directed to the client.

Data is always retrived from the clipboard as a string. When retreiving clipboard data, a data type string speficies the format of the data string. The availability of different clipboard formats is determined by the current clipboard owner.

`get-clipboard-bitmap`

Gets the current clipboard contents as a bitmap (Windows, MacOS), returning `#f` if the clipboard does not contain a bitmap.

- (send *a-clipboard* `get-clipboard-bitmap` *time*) $\Rightarrow$ `bitmap%` object or `#f`
  time : exact integer

  See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`get-clipboard-client`

Gets the current clipboard-owning client, returning `#f` if the clipboard is not owned by a client.

- (send *a-clipboard* `get-clipboard-client`) $\Rightarrow$ `clipboard-client%` object or `#f`

`get-clipboard-data`

Gets the current clipboard contents in a specific format, returning `#f` if the clipboard does not contain data in the requested format.

- (send *a-clipboard* `get-clipboard-data` *format* *time*) $\Rightarrow$ string or `#f`
  format : string
  time : exact integer

  The *format* string is typically four capital letters. (On the Macinotsh, only four characters for *format* are ever used.) For example, `"TEXT"` is the name of the simple text format. New format names can be used to communicate application- and platform-specific data formats.

  See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`get-clipboard-string`

Gets the current clipboard contents as simple text, returning `#f` if the clipboard does not contain any text.

- (send *a-clipboard* `get-clipboard-string` *time*) $\Rightarrow$ string or `#f`
  time : exact integer

See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`set-clipboard-bitmap`

Changes the current clipboard contents to a bitmap (Windows, MacOS).

- (send *a-clipboard* `set-clipboard-bitmap` *new-bitmap time*) ⇒ void
    new-bitmap : `bitmap%` object
    time : exact integer

Sets the clipboard contents to *new-bitmap*. See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`set-clipboard-client`

Changes the clipboard-owning client.

- (send *a-clipboard* `set-clipboard-client` *new-owner time*) ⇒ void
    new-owner : `clipboard-client%` object
    time : exact integer

Sets the client to *new-owner*. See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`set-clipboard-string`

Changes the current clipboard contents to a text string.

- (send *a-clipboard* `set-clipboard-string` *new-text time*) ⇒ void
    new-text : string
    time : exact integer

Sets the clipboard contents to *new-text*. See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

3.12   `clipboard-client%`

A `clipboard-client%` object allows a program to take over the clipboard and service requests for clipboard data. See `clipboard<%>` for more information.

- (make-object `clipboard-client%`) ⇒ `clipboard-client%` object

    Creates a clipboard client that supports no data formats.

`add-type`

Adds a new data format name to the list supported by thw clipboard client.

- (send *a-clipboard-client* `add-type` *format*) ⇒ void
    format : string

The *format* string is typically four capital letters. (On the Macinotsh, only four characters for *format* are ever used.) For example, `"TEXT"` is the name of the simple text format. New format names can be used to communicate application- and platform-specific data formats.

`get-data`

Called when some process requests clipboard data while this client owns the clipboard. The requested format is passed to the method, and the result should be a string matching the requested format, or `#f` if the request cannot be fulfilled. (Only data format names in the client's list will be passed to this method; see also `add-type`.)

- (send *a-clipboard-client* `get-data` *format*) ⇒ string or #f
    format : string

The *format* string is typically four capital letters. (On the Macinotsh, only four characters for *format* are ever used.) For example, `"TEXT"` is the name of the simple text format. New format names can be used to communicate application- and platform-specific data formats.

`get-types`

Returns a list of names that are the data formats supported by the clipboard client.

- (send *a-clipboard-client* `get-types`) ⇒ list of strings

`on-replaced`

Called when a clipboard client is dismissed as the clipboard owner (because the clipboard has be taken by another client or by an external application).

- (send *a-clipboard-client* `on-replaced`) ⇒ void

3.13 `control<%>`

Extends: `subwindow<%>`

The `control<%>` interface is implemented by the built-in control window classes:

- `message%`
- `button%`
- `check-box%`
- `slider%`
- `gauge%`
- `text-field%`
- `radio-box%`
- `choice%`
- `list-box%`

`command`

Calls the control's callback function, passing on the given `control-event%` object.

- (send *a-control* `command` *event*) ⇒ void
    event : `control-event%` object

3.14 `control-event%`

Superclass: `event%`

A `control-event%` object contains information about a control event. An instance of `control-event%` is always provided to a control or menu item callback procedure.

- (make-object `control-event%` *event-type*) ⇒ `control-event%` object
    event-type : symbol   in   '(button check-box choice list-box list-box-dclick
              text-field text-field-enter menu slider radio-box)

  The *event-type* argument is one of the following:

    – 'button — for `button%` clicks
    – 'check-box — for `check-box%` toggles
    – 'choice — for `choice%` item selections
    – 'list-box — for `list-box%` selections and deselections
    – 'list-box-dclick — for `list-box%` double-clicks
    – 'text-field — for `text-field%` changes
    – 'text-field-enter — for single-line `text-field%` Enter event
    – 'menu — for `selectable-menu-item<%>` callbacks
    – 'slider — for `slider%` changes
    – 'radio-box — for `radio-box%` selection changes
    – 'menu-popdown — for `popup-menu%` callbacks (item selected)
    – 'menu-popdown-none — for `popup-menu%` callbacks (no item selected)

  This value is extacted out of a `control-event%` object with the `get-event-type` method.

`get-event-type`

Returns the type of the control event. See `control-event%` for information about each event type symbol.

- (send *a-control-event* `get-event-type`) ⇒ symbol in '(button check-box choice list-box
    list-box-dclick text-field text-field-enter menu slider radio-box)

`set-event-type`

Sets the type of the event. See `control-event%` for information about each event type symbol.

- (send *a-control-event* `set-event-type` *type*) ⇒ void
    type : symbol   in   '(button check-box choice list-box list-box-dclick
          text-field text-field-enter menu slider radio-box)

3.15 `cursor%`

A cursor is a small bitmap that indicates the location of the mouse pointer. The bitmap image typically indicates the current mode or meaning of a mouse click at its current location.

A cursor is assigned to each window (or the window may use its parent's cursor; see `set-cursor` for more information), and the pointer image is changed to match the window's cursor when the pointer is moved over the window. Each cursor object may be assigned to many windows.

- (make-object cursor% *name kind hot-spot-x hot-spot-y*) $\Rightarrow$ cursor% object
    name : string
    kind = 'unknown : symbol in '(unknown gif xbm xpm bmp pict)
    hot-spot-x = 0 : exact integer in [0, 10000]
    hot-spot-y = 0 : exact integer in [0, 10000]

  Creates a cursor using a bitmap. The *name* and *kind* arguments are the same as for `load-file`.

  The *hot-spot-x* and *hot-spot-y* arguments determine the focus point of the cursor within the cursor image, relative to its top-left corner.

  If the cursor is loaded sucessfully, `ok?` returns `#t`, otherwise the cursor object cannot be assigned to a window.

- (make-object cursor% *id*) $\Rightarrow$ cursor% object
    id : symbol in '(arrow bullseye cross hand ibeam watch)

  Creates a cursor using a stock cursor, specified as one of the follwing:

    – 'arrow — the default cursor
    – 'bullseye — concentric circles
    – 'cross — a crosshair
    – 'hand — an open hand
    – 'ibeam — a vertical line, indicating that clicks control a text-selection caret
    – 'watch — a watch or hourglass, indicating that the user must wait for a computation to complete

`ok?`

Returns `#t` if the cursor is can be assigned to a window, `#f` otherwise.

- (send *a-cursor* ok?) $\Rightarrow$ boolean

3.16 `dialog%`

Implements: `top-level-window<%>`

A dialog is a top-level window that is **modal**: while the dialog is shown, all other top-level windows in the dialog's eventspace are disabled.

- (make-object dialog% *label parent width height x y style*) $\Rightarrow$ dialog% object
    label : string
    parent = #f : frame% or dialog% object or #f
    width = #f : exact integer in [0, 10000] or #f
    height = #f : exact integer in [0, 10000] or #f
    x = #f : exact integer in [0, 10000] or #f

$y = $ `#f` : exact integer in [0, 10000] or `#f`

$style = $ `null` : list of symbols in `'(no-caption resize-border)`

The *label* string is used as the dialog's title in its title bar. If the dialog's label is changed (see `set-label`), the title bar is updated.

The *parent* argument can be `#f` or an existing frame. Under Windows, if *parent* is an existing frame, the new dialog is always on top of its parent. Under Windows and X, a dialog is iconized when its parent is iconized.

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

If the *width* or *height* argument is not `#f`, it specifies an initial size for the dialog (in pixles) assuming that it is larger than the minimum size, otherwise the minimum size is used. Under Windows and MacOS (and with some X window managers) dialogs are not resizeable.

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

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

- `'no-caption` — omits the title bar for the dialog (Windows)
- `'resize-border` — adds a resizeable border around the window (Windows) or grow box in the bottom right corner (MacOS)

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

`on-subwindow-char`

Called when this window or a child window receives a keyboard event. The `on-subwindow-char` method of the receiver's top-level window is called first (see `get-top-level-window`); if the return value is `#f`, then the `on-subwindow-char` method is called for the next child in the path to the receiver, and so on. Finally, if the receiver's `on-subwindow-char` method returns `#f`, the event is passed on to the receiver's normal key-handling mechanism.

BEWARE: The default `on-subwindow-char` in `frame%` and `on-subwindow-char` in `dialog%` methods consume certain keyboard events (e.g., arrow keys, Enter) used for navigating within the window. Because the top-level window gets the first chance to handle the keyboard event, some events never reach the "receiver" child unless the default frame or dialog method is overridden.

- (send *a-dialog* `on-subwindow-char` *receiver event*) ⇒ boolean
    receiver : `window<%>` object
    event : `key-event%` object

Returns the result of

    (or (send *this* on-system-menu-char *event*)
        (send *this* on-traverse-char *event*))

`show`

Shows or hides a window.

The visibility of a window can be changed by the user clicking the window's close box, for example, , and such changes do not go through this method; use `on-superwindow-show` or `on-close` to monitor visibility changes.

- (`send` *a-dialog* `show` *show?*) ⇒ void
  show? : boolean

If *show?* is `#f`, the window is hidden. Otherwise, the window is shown.

If the window is already shown, it is moved front of other top-level windows. If the window is iconized (frames only), it is deiconized.

If *show?* is true, the dialog is shown and all frames (and other dialogs) in the eventspace become disabled until the dialog is closed. If *show?* is false, the dialog is hidden and other frames and dialogs are re-enabled (unless a different, pre-existing dialog is still shown).

3.17   `event%`

A `event%` object contains information about a control, keyboard, mouse, or scroll event. See also `control-event%`, `key-event%`, `mouse-event%`, and `scroll-event%`.

`get-time-stamp`

Returns the time, in milliseconds, when the event occurred. This time is compatible with times reported b yMzScheme's `current-milliseconds` procedure.

- (`send` *an-event* `get-time-stamp`) ⇒ exact integer

`set-time-stamp`

Set the time, in milliseconds, when the event occurred. See also MzScheme's `current-milliseconds`.

If the supplied value is outside the platform-specific range of time values, an `exn:application:mismatch` exception is raised.

- (`send` *an-event* `set-time-stamp` *time*) ⇒ void
  time : exact integer

3.18   `frame%`

Implements: `top-level-window<%>`

A frame is a top-level container window. It has a title bar (which displays the frame's label), an optional menu bar, and an optional status line.

Under Windows, both Multiple Document Interface (MDI) and Single Document Interface (SDI) frames are supported.

- (`make-object` `frame%` *label parent width height x y style*) ⇒ `frame%` object
  label : string
  parent = `#f` : `frame%` object or `#f`
  width = `#f` : exact integer in [0, 10000] or `#f`
  height = `#f` : exact integer in [0, 10000] or `#f`
  x = `#f` : exact integer in [0, 10000] or `#f`

y = `#f` : exact integer in [0, 10000] or `#f`

style = `null` : list of symbols in `'(no-resize-border no-caption no-system-menu mdi-parent mdi-child)`

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

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

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

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

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

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

- `'no-resize-border` — omits the resizeable border around the window (Windows) or grow box in the bottom right corner (MacOS)
- `'no-caption` — omits the title bar for the frame (Windows)
- `'no-system-menu` — omits the system menu (Windows)
- `'mdi-child` — creates the frame as a MDI (multiple document interface) child frame, mutually exclsuive with `'mdi-parent` (Windows)
- `'mdi-parent` — creates the frame as a MDI (multiple document interface) parent frame, mutually exclsuive with `'mdi-child` (Windows)

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

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

create-status-line

- (send *a-frame* `create-status-line`) ⇒ void

Creates a status line at the bottom of the frame. The width of the status line is the whole width of the frame (adjusted automatically when resizing), and the height and text size are platform-specific.

See also `set-status-text`.

get-menu-bar

Returns the frame's menu bar, or `#f` if none has been created for the frame.

- (send *a-frame* `get-menu-bar`) ⇒ `menu-bar%` object or `#f`

has-status-line?

Returns `#t` if the frame's status line has been created, `#f` otherwise. See also `create-status-line`.

- (send *a-frame* `has-status-line?`) ⇒ boolean

iconize

Iconizes or deiconizes the frame. Deiconizing brings the frame to the front. Iconization has no effect under MacOS.

A frame's iconization can be changed by the user, and such changes do not go through this method. A program cannot detect when a frame has been iconized except by polling `is-iconized?`.

- (send *a-frame* `iconize` *iconize?*) ⇒ void
    iconize? : boolean

is-iconized?

Returns `#t` if the frame is iconized, `#f` otherwise.

- (send *a-frame* `is-iconized?`) ⇒ boolean

maximize

Maximizes or restores the frame under Windows and MacOS; the frame's show state is not affected. Under Windows, an iconized frame cannot be maximized or restored.

A window's maximization can be changed by the user, and such changes do not go through this method; use `on-size` to monitor size changes.

- (send *a-frame* `maximize` *maximize?*) ⇒ void
    maximize? : boolean

    If *maximize?* is `#f`, the window is restored, otherwise it is maximized.

on-menu-char

If the frame has a menu bar with keyboard shortcuts, `on-menu-char` attempts to match the given event to a menu item. If a match is found, `#t` is returned, otherwise `#f` is returned.

When the match corresponds to a complete shortcut combination, the menu item's callback is called (before `on-menu-char` returns). A match may also correspond to a shortcut prefix (under X, when when prefix style is 'ctl-m; see `set-x-shortcut-prefix`), in which case the prefix key event is consumed and `#t` is returned, but the menu item's callback is not called until the shortcut is completed (if it is completed).

If the event does not correspond to a complete shortcut combination, the event may be handled anyway if it correponds to a mnemonic in the menu bar (i.e., an underlined letter in a menu's title, which is installed by including an ampersand in the menu's label). If a mnemonic match is found, the keyboard focus is moved to the menu bar (selecting the menu with the mnemonic), and `#t` is returned.

- (send *a-frame* `on-menu-char` *event*) ⇒ boolean
    event : `key-event%` object

on-subwindow-char

Called when this window or a child window receives a keyboard event. The `on-subwindow-char` method of the receiver's top-level window is called first (see `get-top-level-window`); if the return value is `#f`, then

the `on-subwindow-char` method is called for the next child in the path to the receiver, and so on. Finally, if the receiver's `on-subwindow-char` method returns `#f`, the event is passed on to the receiver's normal key-handling mechanism.

BEWARE: The default `on-subwindow-char` in `frame%` and `on-subwindow-char` in `dialog%` methods consume certain keyboard events (e.g., arrow keys, Enter) used for navigating within the window. Because the top-level window gets the first chance to handle the keyboard event, some events never reach the "receiver" child unless the default frame or dialog method is overridden.

- (send *a-frame* `on-subwindow-char` *receiver* *event*) $\Rightarrow$ boolean
    receiver : `window<%>` object
    event : `key-event%` object

Returns the result of

```
(or (send this on-menu-char event)
    (send this on-system-menu-char event)
    (send this on-traverse-char event))
```

`set-icon`

Sets the large or small icon bitmap for this frame. Future changes to the bitmap do not affect the frame's icon.

The icon is used in a platform-specific way:

- Windows — the small icon is used for the frame's icon (in the top-left) and in the task bar, and the large icon is used for the Atl-Tab task switcher.

- MacOS — both icons are ignored.

- X — many window managers use the small icon in the same way as Windows, and others use the small icon when iconifying the frame; the large icon is ignored.

The bitmap for either icon can be any size, but most platforms scale the small bitmap to 16 by 16 pixels and the large bitmap to 32 by 32 pixels.

If a mask bitmap is not provided, then the entire (rectangular) bitmap is used as an icon.

If a mask bitmap is provided, the mask must be monochrome. In the mask bitmap, use black pixels to indicate the icon's region and use white pixels outside the icon's region. In the icon bitmap, use black pixels for the region outside the icon.

- (send *a-frame* `set-icon` *icon* *mask* *which*) $\Rightarrow$ void
    icon : `bitmap%` object
    mask = `#f` : `bitmap%` object
    which = `'both` : symbol in `'(small large both)`

`set-status-text`

Sets the frame's status line text and redraws the status line. See also `create-status-line`.

- (send *a-frame* `set-status-text` *text*) $\Rightarrow$ void
    text : string

3.19   gauge%

Implements: `control<%>`

A gauge is a horizontal or vertical bar for displaying the output value of a bounded integer quantity. Each guage has an adjustable range, and the gauge's current value is always between 0 and its range, inclusive. Use `set-value` to set the value of the gauge.

- (make-object gauge% *label range parent style*) ⇒ gauge% object
    label : string or #f
    range : exact integer in [1, 10000]
    parent : frame%, dialog%, panel%, or pane% object
    style = '(horizontal) : list of symbols in '(horizontal vertical)

  If *label* is a string, it is used as the gauge label; otherwise the gauge does not display a label.

  If an ampersand ("&") occurs in *label*, it is specially parsed; under Windows and X, the character following an ampersand is underlined in the displayed control to indicate a keyboard mnemonic. (Under MacOS, mnemonic underlines are not shown.) The mnemonic is meaningless for a gauge (as far as `on-traverse-char` is concerned), but it is supported for consistentcy with other control types. A programmer may assign a meaning to the mneumonic, e.g., by overriding `on-traverse-char`.

  The *range* argument is an integer specifying the maximum value of the gauge (inclusive). The minimum guage value is always `0`.

  The *style* list must include either '`horizontal`, specifying a horizontal gauge, or '`vertical`, specifying a vertical gauge.

`get-range`

Returns the range (maximum value) of the gauge.

- (send *a-gauge* get-range) ⇒ exact integer in [1, 10000]

`get-value`

Returns the gauge's current value.

- (send *a-gauge* get-value) ⇒ exact integer in [0, 10000]

`set-range`

Sets the range (maximum value) of the gauge.

- (send *a-gauge* set-range *range*) ⇒ void
    range : exact integer in [1, 10000]

`set-value`

Sets the gauge's current value. If the specified value is larger than the gauge's range, an `exn:application:mismatch` exception is raised.

- (send *a-gauge* `set-value` *pos*) ⇒ void
   pos : exact integer in [0, 10000]

3.20   grow-box-spacer-pane%

Superclass: `pane%`

See `pane%`.

- (make-object grow-box-spacer-pane% *parent*) ⇒ grow-box-spacer-pane% object
   parent : `frame%`, `dialog%`, `panel%`, or `pane%` object

3.21   horizontal-pane%

Superclass: `pane%`

A horizontal pane arranges its subwindows in a single row. See also `pane%`.

- (make-object horizontal-pane% *parent*) ⇒ horizontal-pane% object
   parent : `frame%`, `dialog%`, `panel%`, or `pane%` object

3.22   horizontal-panel%

Superclass: `panel%`

A horizontal panel arranges its subwindows in a single row. See also `panel%`.

- (make-object horizontal-panel% *parent* *style*) ⇒ horizontal-panel% object
   parent : `frame%`, `dialog%`, `panel%`, or `pane%` object
   style = `null` : list of symbols in `'(border)`

   If the `'border` style is specified, the window is created with a thin border (only in this case, the client size of the panel may be less than its total size).

3.23   key-event%

Superclass: `event%`

A `key-event%` object contains information about a key event. Key events are primarily processed by `on-subwindow-char` in `window<%>` and `on-char` in `canvas<%>`.

See also section 2.3 (page 12).

- (make-object key-event%) ⇒ key-event% object

`get-alt-down`

Returns `#t` if the Option (MacOS) key was down for the event. When the Alt key is pressed in Windows, it is reported as a Meta press (see `get-meta-down`).

- (send *a-key-event* get-alt-down) ⇒ boolean

get-control-down

Returns #t if the Control key was down for the event.

- (send *a-key-event* get-control-down) ⇒ boolean

get-key-code

Gets the virtual key code for the key event. The virtual key code is either a character or a special key symbol, one of the following:

- 'start
- 'cancel
- 'clear
- 'shift
- 'control
- 'menu
- 'pause
- 'capital
- 'prior
- 'next
- 'end
- 'home
- 'left
- 'up
- 'right
- 'down
- 'select
- 'print
- 'execute
- 'snapshot
- 'insert
- 'help
- 'numpad0
- 'numpad1
- 'numpad2
- 'numpad3
- 'numpad4
- 'numpad5
- 'numpad6
- 'numpad7
- 'numpad8
- 'numpad9
- 'multiply
- 'add
- 'separator
- 'subtract
- 'decimal
- 'divide
- 'f1

- 'f2
- 'f3
- 'f4
- 'f5
- 'f6
- 'f7
- 'f8
- 'f9
- 'f10
- 'f11
- 'f12
- 'f13
- 'f14
- 'f15
- 'f16
- 'f17
- 'f18
- 'f19
- 'f20
- 'f21
- 'f22
- 'f23
- 'f24
- 'numlock
- 'scroll

The special key symbols attempt to capture useful keys that have no standard ASCII representation. A few keys have standard reprsentations that are not obvious:

- `#\space` — the space bar

- `#\return` — the Enter or Return key (on all platforms), but not necessarily the Enter key near the numpad (which is reported as `'numpad-enter` if the platform distinguishes the two Enter keys)

- `#\tab` — the tab key

- `#\backspace` — the backspace key

- `#\rubout` — the delete key

If a suitable special key symbol or ASCII representation is not available, `#\nul` (the null character) is reported.

- (send *a-key-event* `get-key-code`) ⇒ character or symbol

`get-meta-down`

Returns `#t` if the Meta (X), Alt (Windows), or Command (MacOS) key was down for the event.

Under MacOS, if a command-key press is combined with a mouse button click, the event is reported as a right-button click and `get-meta-down` for the event reports `#f`.

- (send *a-key-event* `get-meta-down`) ⇒ boolean

`get-shift-down`

Returns `#t` if the Shift key was down for the event.

- (send *a-key-event* `get-shift-down`) $\Rightarrow$ boolean

`get-x`

Returns the x-position of the mouse at the time of the event, in the target's window's (client-area) coordinate system.

- (send *a-key-event* `get-x`) $\Rightarrow$ real number

`get-y`

Returns the y-position of the mouse at the time of the event in the target's window's (client-area) coordinate system.

- (send *a-key-event* `get-y`) $\Rightarrow$ real number

`set-alt-down`

Sets whether the Option (MacOS) key was down for the event. When the Alt key is pressed in Windows, it is reported as a Meta press (see `set-meta-down`).

- (send *a-key-event* `set-alt-down` *down?*) $\Rightarrow$ void
     down? : boolean

`set-control-down`

Sets whether the Control key was down for the event.

- (send *a-key-event* `set-control-down` *down?*) $\Rightarrow$ void
     down? : boolean

`set-key-code`

Sets the virtual key code for the event, either a character or one of the special symbols listed with `get-key-code`.

- (send *a-key-event* `set-key-code` *code*) $\Rightarrow$ void
     code : character or symbol

`set-meta-down`

Sets whether the Meta (X), Alt (Windows), or Command (MacOS) key was down for the event.

Under MacOS, if a command-key press is combined with a mouse button click, the event is reported as a right-button click and `get-meta-down` for the event reports `#f`.

  - (send *a-key-event* `set-meta-down` *down?*) ⇒ void
      down? : boolean

set-shift-down

Sets whether the Shift key was down for the event.

  - (send *a-key-event* `set-shift-down` *down?*) ⇒ void
      down? : boolean

set-x

Sets the x-position of the mouse at the time of the event in the target's window's (client-area) coordinate system.

  - (send *a-key-event* `set-x` *pos*) ⇒ void
      pos : real number

set-y

Sets the y-position of the mouse at the time of the event in the target's window's (client-area) coordinate system.

  - (send *a-key-event* `set-y` *pos*) ⇒ void
      pos : real number

3.24 `labelled-menu-item<%>`

Extends: `menu-item<%>`

A `labelled-menu-item<%>` object is a `menu-item<%>` with a string label (i.e., any menu item other than a separator). More specifically, it is an instance of either `menu-item%` (a plain menu item), `checkable-menu-item%` (a checkable menu item), or `menu%` (a submenu).

enable

Enables or disables the menu item. If the item is a submenu (or menu in a menu bar), the entire menu is disabled, but each submenu item's `is-enabled?` method returns `#f` only if the item is specifically disabled (in addition to the submenu).

  - (send *a-labelled-menu-item* `enable` *enabled?*) ⇒ void
      enabled? : boolean

get-help-string

Returns the help string for the menu item, or `#f` if the item has no help string.

When an item has a *help-string*, the string may be used to display help information to the user.

- (send *a-labelled-menu-item* get-help-string) ⇒ string or #f

get-label

Returns the item's label.

See also `set-label` and `get-plain-label`.

- (send *a-labelled-menu-item* get-label) ⇒ string

get-plain-label

Like `get-label`, except that ampersands in the label are removed as described in `set-label`.

- (send *a-labelled-menu-item* get-plain-label) ⇒ string

is-enabled?

Returns #t if the menu item is enabled, #f otherwise.

See also `enable`.

- (send *a-labelled-menu-item* is-enabled?) ⇒ boolean

on-demand

Normally called when the user clicks on the menu bar containing the item (before the user sees any menu items), just before the popup menu containing the item is popped up, or just before inspecting the menu bar containing the item for a shortcut key binding.

A `on-demand` in `menu-item-container<%>` method can be overridden in such a way that the container does not call the `on-demand` method of its items.

- (send *a-labelled-menu-item* on-demand) ⇒ void

set-help-string

Sets the help string for the menu item. Use #f to remove the help string for an item.

- (send *a-labelled-menu-item* set-help-string *help*) ⇒ void
    help : string or #f

set-label

Sets the menu item's label. If the item has a shortcut, the shortcut is not affected.

If the label contains an ampersand ("&") and the window is a control, the label is parsed specially; under Windows, the character following an ampersand is underlined in the displayed menu to indicate a keyboard mnemonic. Pressing and releasing the Alt key switches to menu-selection mode in the menu bar where

mnemonic characters are used for navigation. (An Alt combination might select a menu via `on-menu-char`.) A double-ampersand in the label is replaced by a literal (non-navigation) ampersand. Under X and MacOS, ampersands in the label are parsed in the same way as for Windows, but no mnemonic underline is displayed.

An ampersand is always preserved in the label returned by `get-label`, but never preserved in the label returned by `get-plain-label`.

- (send *a-labelled-menu-item* `set-label` *label*) ⇒ void
    label : string

3.25   `list-box%`

Implements: `list-control<%>`

A list box allows the user to select one or more string items from a scrolling list. A list box is either a single-selection control (if an item is selected, the previous selection is removed) or a multiple-selection control (clicking an item toggles the item on or off independently of other selections).

Whenever the user changes the selection in a list box, the list box's callback procedure is called. A callback procedure is provided as an initialization argument when each list box is created.

List box items are indexed from `0`.

See also `choice%`.

- (make-object `list-box%` *label choices parent callback style*) ⇒ `list-box%` object
    label : string or #f
    choices : list of strings
    parent : frame%, dialog%, panel%, or pane% object
    callback : procedure of two arguments: a `list-box%` object and a `control-event%` object
    style = '(single) : list of symbols in '(single multiple extended)

  If *label* is not `#f`, it is used as the list box label. Otherwise, the list box will not display its label.

  If an ampersand ("&") occurs in *label*, it is specially parsed; under Windows and X, the character following an ampersand is underlined in the displayed control to indicate a keyboard mnemonic. (Under MacOS, mnemonic underlines are not shown.) The underlined mnemonic character must be a letter or a digit. The user can move the keyboard focus to the list box by typing the mnemonic when the control's top-level-window contains the keyboard focus. The user must also hold down the Meta or Alt key if the keyboard focus is currently in a control that handles normal alphanumeric input. The ampersand itself is removed from *label* before it is displayed for the control; a double-ampersand in *label* is converted to a single ampersand (with no mnemonic underlining). Mnemonic keyboard events are handled by `on-traverse-char` (but not under MacOS).

  The *choices* list specifies the initial list of items to appear in the list box.

  The *callback* procedure is called when the user changes the list box selection, by either selecting, re-selecting, deselecting, or double-clicking an item. The type of the even provided to the callback is `'list-box-dclick` when the user double-clicks on an item, or `'list-box` otherwise.

  The *style* specification must include exactly one of the following:

    – `'single` — Creates a single-selection list.
    – `'multiple` — Creates a multiple-selection list where a single click deselects other items and selects a new item. Use this style for a list when single-selection is common, but multiple selections are allowed.

– `'extended` — Creates a multiple-selection list where a single click extends the selection. Use this style for a list when multiple selections are the rule rather than the exception.

The `'multiple` and `'extended` styles determine a platform-independent interpretation of unmodified mouse clicks, but dragging, shift-clicking, control-clicking, etc. have platform-standard interpretations. Whatever the platform-specific interface, the user can always select disjoint sets of items or deselect items (and leave no items selected). On some platforms, the user can deselect the (sole) selected item in a `'single` list box.

In addition to the selection style, the *style* list can include one or more of the following:

– `'always-sb` — Creates the vertical scrollbar immediately (otherwise, in Windows, the scrollbar is not created until it is necessary).
– `'hscroll` — Creates a horizontal scrollbar if the item strings are two wide (otherwise the string items are clipped).

append

Adds a new item to the list of user-selectable items. The current selection is unchanged (unless the list control is an empty choice control, in which case the new item is selected).

- (send *a-list-box* `append` *item data*) ⇒ void
    item : string
    data : value

Adds a new item to the list box with an associated "data" object. The *data* object is not displayed in the list box; it is provided merely as a convenience for use with `get-data`, possibly allowing a programmer to avoid managing a separate item-to-data mapping in addition to the list box control.

- (send *a-list-box* `append` *item*) ⇒ void
    item : string

delete

Deletes a choice from the list box. Selected items that are not deleted remain selected, and no other items are selected.

- (send *a-list-box* `delete` *n*) ⇒ void
    n : exact non-negative integer

Deletes the item indexed by *n*. List box items are indexed from 0. If *n* is equal to or larger than the number of items in the control, an `exn:application:mismatch` exception is raised.

get-data

Returns the data value associated with a list box item, or `#f` if there is no assciated data. See also `append` and `set-data`.

- (send *a-list-box* `get-data` *n*) ⇒ value
    n : exact non-negative integer

Returns the data for the item indexed by *n*. List box items are indexed from 0. If *n* is equal to or larger than the number of choices, an `exn:application:mismatch` exception is raised.

`get-first-visible-item`

Reports the index of the item currently scrolled to the top of the list box. List box items are indexed from 0.

- (send *a-list-box* `get-first-visible-item`) ⇒ exact non-negative integer

`get-selections`

Returns a list of indicies for all currently selected items. List box items are indexed from 0.

For single-selection lists, the result is always either `null` or a list containing one number.

- (send *a-list-box* `get-selections`) ⇒ list of exact integers

`is-selected?`

Returns `#t` if the item matching the specifies index is selected, `#f` otherwise.

A list box's selection can be changed by the user clicking the control, and such changes do not go through this method; use the control callback procedure (provided as an initialization argument) to monitor selection changes.

- (send *a-list-box* `is-selected?`   *n*) ⇒ boolean
    n : exact non-negative integer

    Returns `#t` if the item index by *n* is selected. List box items are indexed from 0. If *n* is equal to or larger than the number of choices, an `exn:application:mismatch` exception is raised.

`number-of-visible-items`

Returns the maximum number of items in the list box that are visible to the user with the control's current size (rounding down if the exact answer is fractional, but returning at least 1).

- (send *a-list-box* `number-of-visible-items`) ⇒ exact positive integer

`select`

Selects or deselects a item. For selection in a single-selection list box, if a different choice is currently selected, it is automatically deselected. For selection in a multiple-selection list box, other selections are preserved, unlike `set-selection`.

A list box's selection can be changed by the user clicking the control, and such changes do not go through this method; use the control callback procedure (provided as an initialization argument) to monitor selection changes.

The control's callback procedure is *not* invoked.

- (send *a-list-box* `select` *n* *select?*) ⇒ void
    n : exact non-negative integer
    select? = `#t` : boolean

If *select?* is #f, the item indexed by n is deselected; otherwise it is selected. List box items are indexed from 0. If n is equal to or larger than the number of choices, an `exn:application:mismatch` exception is raised.

set

Clears the list box and installs a new list of items.

- (send *a-list-box* set *choices*) ⇒ void
  choices : list of strings

set-data

Sets the associated data for a list box choice item. See also `append`.

- (send *a-list-box* set-data *n* *data*) ⇒ void
  n : exact non-negative integer
  data : value

Sets the associated data for item indexed by n. List box items are indexed from 0. If n is equal to or larger than the number of choices, an `exn:application:mismatch` exception is raised.

set-first-visible-item

Scrolls the list box so that the specified item is at the top of the list box display.

A list box's scroll position can be changed by the user clicking the control, and such changes do not go through this method. A program cannot detect when the scroll position changes except by polling `get-first-visible-item`.

- (send *a-list-box* set-first-visible-item *n*) ⇒ void
  n : exact non-negative integer

Shows the item indexed by n at the list box's top. List box items are indexed from 0. If n is equal to or larger than the number of choices, an `exn:application:mismatch` exception is raised.

set-string

Changes an item in the list box.

- (send *a-list-box* set-string *n* *label*) ⇒ void
  n : exact non-negative integer
  label : string

Sets the item indexed by n. List box items are indexed from 0. If n is equal to or larger than the number of choices, an `exn:application:mismatch` exception is raised.

3.26 `list-control<%>`

Extends: `control<%>`

A list control gives the user a list of string items to choose from. There are two built-in classes that implement `list-control<%>`:

- `choice%` — presents the list in a popup menu (so the user can choose only one item at a time)

- `list-box%` — presents the list in a scrolling box, allowing the use to choose one item (if the style includes `'single`) or any number of items

In either case, the set of user-selectable items can be changed dynamically.

append

Adds a new item to the list of user-selectable items. The current selection is unchanged (unless the list control is an empty choice control, in which case the new item is selected).

- (`send` *a-list-control* `append` *item*) $\Rightarrow$ void
    item : string

clear

Removes all user-selectable items from the control.

- (`send` *a-list-control* `clear`) $\Rightarrow$ void

find-string

Finds a user-selectable item matching the given string. If no matching choice is found, `#f` is returned, otherwise the index of the matching choice is returned (items are indexed from `0`).

- (`send` *a-list-control* `find-string` *s*) $\Rightarrow$ exact non-negative integer or `#f`
    s : string

get-number

Returns the number of user-selectable items in the control (which is also one more than the greatest index in the list control).

- (`send` *a-list-control* `get-number`) $\Rightarrow$ exact non-negative integer

get-selection

Returns the index of the currently selected item (items are indexed from `0`). If the choice item currently contains no choices or no selections, `#f` is returned. If multiple selections are allowed and multiple items are selected, the index of the first selection is returned.

- (`send` *a-list-control* `get-selection`) $\Rightarrow$ exact non-negative integer or `#f`

`get-string`

Returns the item for the given index (items are indexed from `0`). If the provided index is larger than the greatest index in the list control, an `exn:application:mismatch` exception is raised.

- (send *a-list-control* `get-string` *n*) ⇒ string or `#f`
    n : exact non-negative integer

`get-string-selection`

Returns the currently selected item. If the control currently contains no choices, `#f` is returned. If multiple selections are allowed and multiple items are selected, the first selection is returned.

- (send *a-list-control* `get-string-selection`) ⇒ string or `#f`

`set-selection`

Selects the item specified by the given index (items are indexed from `0`). If the given index larger than the greatest index in the list control, an `exn:application:mismatch` exception is raised.

In a list box control, all other items are deselected, even if multiple selections are allowed in the control. See also `select` in `list-box%`.

The control's callback procedure is *not* invoked when this method is called.

The list control's selection can be changed by the user clicking the control, and such changes do not go through this method; use the control callback procedure (provided as an initialization argument) to monitor selection changes.

- (send *a-list-control* `set-selection` *n*) ⇒ void
    n : exact non-negative integer

`set-string-selection`

Selects the item matching that matches the given string. If no match is found in the list control, an `exn:application:mismatch` exception is raised.

In a list box control, all oter items are deselected, even if multiple selections are allowed in the control. See also `select` in `list-box%`.

The control's callback procedure is *not* invoked when this method is called.

The list control's selection can be changed by the user clicking the control, and such changes do not go through this method; use the control callback procedure (provided as an initialization argument) to monitor selection changes.

- (send *a-list-control* `set-string-selection` *s*) ⇒ void
    s : string

3.27   menu%

Implements: `labelled-menu-item<%>`

Implements: `menu-item-container<%>`

A `menu%` object is a submenu within a `menu%` or `popup-menu%`, or as a top-level menu in a `menu-bar%`.

- (make-object menu% *label* *parent* *help*) ⇒ menu% object
    label : string
    parent : menu%, popup-menu%, or menu-bar% object
    help = #f : string or #f

Creates a new menu with the given label.

If *label* contains an ampersand ("&"), it is handled specially; under Windows, the character following an ampersand is underlined in the displayed menu title to indicate a keyboard mnemonic. Pressing and releasing the Alt key switches to menu-selection mode in the menu bar where mnemonic characters are used for navigation. An Alt combination might select a specific menu via `on-menu-char`. A double-ampersand in *label* is replaced by a literal (non-navigation) ampersand. Under X and MacOS, ampersands in the label are parsed in the same way as for Windows, but no mnemonic underline is displayed.

If help is not `#f`, the menu has a help string. See `get-help-string` for more information.

If the menu has the label "Help" in a menu bar, it is treated specially on some platforms. Under X, the "Help" menu is typically right-aligned in the menu bar (sometimes only if it is the last menu). Under MacOS, the items of a "Help" menu are folded into the standard help menu. In addition, under MacOS, if the name of the first item in the "Help" menu starts with "About", then the menu item is duplicated as the first item under the Apple menu.

3.28   menu-bar%

Implements: `menu-item-container<%>`

A `menu-bar%` object is created for a particular `frame%` object. A frame can have at most one menu bar; an `exn:application:mismatch` exception is raised when a new menu bar is created for a frame that already has a menu bar.

- (make-object menu-bar% *frame*) ⇒ menu-bar% object
    frame : frame% object

Creates a menu bar in the specified frame. The menu bar is initially empty.

enable

Enables or disables the menu bar (i.e., all of its menus). Each menu's `is-enabled?` method returns `#f` only if the menu is specifically disabled (in addition to the menu bar).

- (send *a-menu-bar* enable *enable?*) ⇒ void
    enable? : boolean

`get-frame`

Returns the menu bar's frame.

- (send *a-menu-bar* `get-frame`) $\Rightarrow$ `frame%` object

`is-enabled?`

Returns `#t` if the menu bar is enabled, `#f` otherwise.

- (send *a-menu-bar* `is-enabled?`) $\Rightarrow$ boolean

3.29 `menu-item<%>`

A `menu-item<%>` object is an element within a `menu%`, `popup-menu%`, or `menu-bar%`. Operations that affect the parent — such as renaming the item, deleting the item, or adding a check beside the item — are accomplished via the `menu-item<%>` object.

A menu item is either a `separator-menu-item%` object (merely a separator), of an `labelled-menu-item<%>` object; the latter is more specifically an instance of either `menu-item%` (a plain menu item), `checkable-menu-item%` (a checkable menu item), or `menu%` (a submenu).

`delete`

Removes the item from its parent. If the menu item is already deleted, `delete` has no effect.

See also `restore`.

- (send *a-menu-item* `delete`) $\Rightarrow$ void

`get-parent`

Returns the menu, popup menu, or menu bar containing the item. The parent for a menu item is specified when the menu item is created, and it cannot be changed.

- (send *a-menu-item* `get-parent`) $\Rightarrow$ `menu%`, `popup-menu%`, or `menu-bar%` object

`is-deleted?`

Returns `#t` if the menu item is deleted from its parent, `#f` otherwise.

- (send *a-menu-item* `is-deleted?`) $\Rightarrow$ boolean

`restore`

Adds a deleted item back into its parent. The item is always restored to the end of the parent, regardless of its original position. If the item is not currently deleted, `restore` has no effect.

- (send *a-menu-item* `restore`) $\Rightarrow$ void

3.30   `menu-item%`

Implements: `selectable-menu-item<%>`

A `menu-item%` is a plain string-labelled menu item. Its parent must be a `menu%` or `popup-menu%`. When the user selects the menu item, its callback procedure is called.

  - (`make-object menu-item%` *label parent callback shortcut help*) ⇒ `menu-item%` object
      label : string
      parent : `menu%` or `popup-menu%` object
      callback : procedure of two arguments: a `menu-item%` object and a `control-event%` object
      shortcut = `#f` : character or `#f`
      help = `#f` : string or `#f`

  Creates a new menu item in *parent*. The item is initially shown, appended to the end of its parent. The *callback* procedure is called (with the event type `'menu`) when the user selects the menu item (either via a menu bar or `popup-menu` in `canvas<%>`).

  See `set-label` for information about mnemonic ampersands ("&") in *label*.

  If shortcut is not `#f`, the item has a shortcut. See `get-shortcut` for more information.

  If help is not `#f`, the item has a help string. See `get-help-string` for more information.

3.31   `menu-item-container<%>`

A `menu-item-container<%>` object is a `menu%`, `popup-menu%`, or `menu-bar%`.

`get-items`

Returns a list of the items in the menu, popup menu, or menu bar. The order of the items in the returned list corresponds to the order as the user sees them in the menu or menu bar.

  - (`send` *a-menu-item-container* `get-items`) ⇒ list of `menu-item<%>` objects

`on-demand`

Called when the user clicks on the container as a menu bar (before the user sees any menu items), just before the container a as popup menu is popped up, or just before inspecting the menu bar containing the item for a shortcut key binding.

If the container is not a menu bar or a popup menu, this method is normally called via the `on-demand` method of the container's owning menu bar or popup menu, because the default implementation of the method chains to the `on-demand` method of its items. However, the method can be overridden in a container such that it does not call the `on-demand` method of its items.

  - (`send` *a-menu-item-container* `on-demand`) ⇒ void

3.32   `message%`

Implements: `control<%>`

A message control is a static line of text or a static bitmap. The text or bitmap corresponds to the message's label (see `set-label`).

- (make-object message% *label* *parent* *style*) ⇒ message% object
   label : string or bitmap% object
   parent : frame%, dialog%, panel%, or pane% object
   style = null : an empty list of symbols

   Creates a string or bitmap message initally showing *message*. If *message* is a bitmap, then the bitmap must be valid (see `ok?` in `bitmap%`) and not installed in a `bitmap-dc%` object; otherwise, an `exn:application:mismatch` exception is raised.

   If an ampersand ("&") occurs in *label*, it is specially parsed; under Windows and X, the character following an ampersand is underlined in the displayed control to indicate a keyboard mnemonic. (Under MacOS, mnemonic underlines are not shown.) The mnemonic is meaningless for a message (as far as `on-traverse-char` is concerned), but it is supported for consistentcy with other control types. A programmer may assign a meaning to the mneumonic, e.g., by overriding `on-traverse-char`.

   The *style* argument is provided for future extensions. Currently, *style* must be the empty list.

`set-label`

Sets a window's label. The window's natural minimum size might be different after the label is changed, but the window's mininum size is not recomputed.

See `get-label` for more information.

- (send *a-message* set-label *label*) ⇒ void
   label : bitmap% object

   Sets the bitmap label for a bitmap message. Since *label* is a bitmap, the bitmap must be valid (see `ok?` in `bitmap%`) and not installed in a `bitmap-dc%` object; otherwise, an `exn:application:mismatch` exception is raised. The bitmap label is installed only if the control was originally created with a bitmap label.

- (send *a-message* set-label *l*) ⇒ void
   l : string or #f

   If *l* is #f, the window's label is removed.

3.33 `mouse-event%`

Superclass: `event%`

A `mouse-event%` object encapsulates a mouse event. Mouse events are primarily processed by `on-subwindow-event` in `window<%>` and `on-event` in `canvas<%>`.

See also section 2.3 (page 12).

- (make-object mouse-event% *event-type*) ⇒ mouse-event% object
   event-type : symbol   in   '(enter leave left-down left-up middle-down middle-up
               right-down right-up motion)

   Creates a mouse event for a particular type of event. The event types are:

   − 'enter — mouse pointer entered the window

      – `'leave` — mouse pointer left the window
      – `'left-down` — left mouse button pressed
      – `'left-up` — left mouse button released
      – `'middle-down` — middle mouse button pressed
      – `'middle-up` — middle mouse button released
      – `'right-down` — right mouse button pressed (MacOS: click with command key pressed)
      – `'right-up` — right mouse button released (MacOS: release with command key pressed)
      – `'motion` — mouse moved, with or without button(s) pressed

`button-changed?`

Returns `#t` if this was a mouse button press or release event, `#f` otherwise. See also `button-up?` and `button-down?`.

  - (send *a-mouse-event* `button-changed?` *button*) ⇒ boolean
     button = `'any` : symbol in `'(left middle right any)`

    If *button* is not `'any`, then `#t` is only returned if it is a release event for a specific button.

`button-down?`

Returns `#t` if the event is for a button press, `#f` otherwise.

  - (send *a-mouse-event* `button-down?` *button*) ⇒ boolean
     button = `'any` : symbol in `'(left middle right any)`

    If *button* is not `'any`, then `#t` is only returned if it is a press event for a specific button.

`button-up?`

Returns `#t` if the event is for a button release, `#f` otherwise.

  - (send *a-mouse-event* `button-up?` *button*) ⇒ boolean
     button = `'any` : symbol in `'(left middle right any)`

    If *button* is not `'any`, then `#t` is only returned if it is a release event for a specific button.

`dragging?`

Returns `#t` if this was a dragging event (motion while a button is pressed), `#f` otherwise.

  - (send *a-mouse-event* `dragging?`) ⇒ boolean

`entering?`

Returns `#t` if this event is for the mouse entering a window, `#f` otherwise.

  - (send *a-mouse-event* `entering?`) ⇒ boolean

`get-alt-down`

Returns `#t` if the Option (MacOS) key was down for the event. When the Alt key is pressed in Windows, it is reported as a Meta press (see `get-meta-down`).

- (send *a-mouse-event* `get-alt-down`) ⇒ boolean

`get-control-down`

Returns `#t` if the Control key was down for the event.

- (send *a-mouse-event* `get-control-down`) ⇒ boolean

`get-event-type`

Returns the type of the event; see `mouse-event%` for information about each event type. See also `set-event-type` .

- (send *a-mouse-event* `get-event-type`) ⇒ symbol in `'(enter leave left-down left-up middle-down middle-up right-down right-up motion)`

`get-left-down`

Returns `#t` if the left mouse button was down (but not pressed) during the event.

- (send *a-mouse-event* `get-left-down`) ⇒ boolean

`get-meta-down`

Returns `#t` if the Meta (X), Alt (Windows), or Command (MacOS) key was down for the event.

Under MacOS, if a command-key press is combined with a mouse button click, the event is reported as a right-button click and `get-meta-down` for the event reports `#f`.

- (send *a-mouse-event* `get-meta-down`) ⇒ boolean

`get-middle-down`

Returns `#t` if the middle mouse button was down (but not pressed) for the event. Under MacOS, a middle-button click is impossible.

- (send *a-mouse-event* `get-middle-down`) ⇒ boolean

`get-right-down`

Returns `#t` if the right mouse button was down (but not pressed) for the event. Under MacOS, a command-click combination is treated as a right-button click.

- (send *a-mouse-event* `get-right-down`) ⇒ boolean

`get-shift-down`

Returns `#t` if the Shift key was down for the event.

- `(send` *a-mouse-event* `get-shift-down)` ⇒ boolean

`get-x`

Returns the x-position of the mouse at the time of the event, in the target's window's (client-area) coordinate system.

- `(send` *a-mouse-event* `get-x)` ⇒ real number

`get-y`

Returns the y-position of the mouse at the time of the event in the target's window's (client-area) coordinate system.

- `(send` *a-mouse-event* `get-y)` ⇒ real number

`leaving?`

Returns `#t` if this event is for the mouse leaving a window, `#f` otherwise.

- `(send` *a-mouse-event* `leaving?)` ⇒ boolean

`moving?`

Returns `#t` if this was a moving event (motion while no button is pressed), `#f` otherwise.

- `(send` *a-mouse-event* `moving?)` ⇒ boolean

`set-alt-down`

Sets whether the Option (MacOS) key was down for the event. When the Alt key is pressed in Windows, it is reported as a Meta press (see `set-meta-down`).

- `(send` *a-mouse-event* `set-alt-down` *down?*`)` ⇒ void
     down? : boolean

`set-control-down`

Sets whether the Control key was down for the event.

- `(send` *a-mouse-event* `set-control-down` *down?*`)` ⇒ void
     down? : boolean

`set-event-type`

Sets the type of the event; see `mouse-event%` for information about each event type. See also `get-event-type`
.

- (send *a-mouse-event* `set-event-type` *event-type*) ⇒ void
    event-type : symbol   in   '(enter leave left-down left-up middle-down middle-up
                    right-down right-up motion)

`set-left-down`

Sets whether the left mouse button was down (but not pressed) during the event.

- (send *a-mouse-event* `set-left-down` *down?*) ⇒ void
    down? : boolean

`set-meta-down`

Sets whether the Meta (X), Alt (Windows), or Command (MacOS) key was down for the event.

Under MacOS, if a command-key press is combined with a mouse button click, the event is reported as a
right-button click and `get-meta-down` for the event reports `#f`.

- (send *a-mouse-event* `set-meta-down` *down?*) ⇒ void
    down? : boolean

`set-middle-down`

Sets whether the middle mouse button was down (but not pressed) for the event. Under MacOS, a middle-
button click is impossible.

- (send *a-mouse-event* `set-middle-down` *down?*) ⇒ void
    down? : boolean

`set-right-down`

Sets whether the right mouse button was down (but not pressed) for the event. Under MacOS, a command-
click combination by the user is treated as a right-button click.

- (send *a-mouse-event* `set-right-down` *down?*) ⇒ void
    down? : boolean

`set-shift-down`

Sets whether the Shift key was down for the event.

- (send *a-mouse-event* `set-shift-down` *down?*) ⇒ void
    down? : boolean

`set-x`

Sets the x-position of the mouse at the time of the event in the target's window's (client-area) coordinate system.

- (send *a-mouse-event* `set-x` *pos*) $\Rightarrow$ void
  pos : real number

`set-y`

Sets the y-position of the mouse at the time of the event in the target's window's (client-area) coordinate system.

- (send *a-mouse-event* `set-y` *pos*) $\Rightarrow$ void
  pos : real number

3.34   `pane%`

Implements: `area-container<%>`

Implements: `subarea<%>`

A pane is a both a container and a containee area. It serves only as a geometry management device. A `pane%` cannot be hidden or disabled like a `panel%` object.

A `pane%` object has a degenerate placement strategy for managing its children; it places them all in the upper left corner and does not stretch any of them. The `horizontal-pane%` and `vertical-pane%` classes provide useful geometry management.

The `grow-box-spacer-pane%` is intended for use as a lightweight spacer in the bottom-right corner of a frame, rather than as a container. Under MacOS, a `grow-box-spacer-pane%` has the same width and height as the grow box that is inset into the bottom-right corner of a frame. Under Windows and X, a `grow-box-spacer-pane%` has zero width and height. Unlike all other container types, a `grow-box-spacer-pane%` is unstretchable by default.

- (make-object `pane%` *parent*) $\Rightarrow$ `pane%` object
  parent : `frame%`, `dialog%`, `panel%`, or `pane%` object

3.35   `panel%`

Implements: `area-container-window<%>`

Implements: `subwindow<%>`

A panel is a both a container and a containee window. It serves mainly as a geometry management device, but the `'border` creates a container with a border. Unlike a `pane%` object, a `panel%` object can be hidden or disabled.

A `panel%` object has a degenerate placement strategy for managing its children; it places them all in the upper left corner and does not stretch any of them. The `horizontal-panel%` and `vertical-panel%` classes provide useful geometry management.

- (make-object panel% *parent* *style*) ⇒ panel% object
   parent : `frame%`, `dialog%`, `panel%`, or `pane%` object
   style = `null` : list of symbols in '(border)

   If the '`border` style is specified, the window is created with a thin border (only in this case, the client size of the panel may be less than its total size).

3.36   `popup-menu%`

Implements: `menu-item-container<%>`

A `popup-menu%` object is created without a parent. Dynamically display a `popup-menu%` with `popup-menu` in `canvas<%>`.

A popup menu is *not* a control. A `choice%` control, however, displays a single value that the user selects from a popup menu. A `choice%` control's popup menu is built into the control, and it is not accessible to the programmer.

- (make-object popup-menu% *title* *callback*) ⇒ popup-menu% object
   title = `#f` : string or #f
   callback = `(lambda (m e) (void))` : procedure of two arguments: a popup-menu% object and a control-event% obje

   If *title* is not `#f`, it is used as a displayed title at the top of the popup menu.

   If *title* contains an ampersand ("&"), it is handled specially, the same as for `menu%` titles. A popup menu mnemonic is not useful, but it is supported for consistentcy with other menu labels.

   The *callback* procedure is invoked when a popup menu is dismissed. If the popup menu is dismissed without an item being selected, *callback* is given a `control-event%` object with the event type '`menu-popdown-none`. If the popup menu is dismissed via an item selection, the item's callback is invoked first, and then *callback* is given a `control-event%` object with the event type '`menu-popdown`.

get-popup-target

Returns the context in which the popup menu is currently displayed, or `#f` if it is not popped up in any window.

The context is set before the `on-demand` method is called, and it is not removed until after the popup-menu's callback is invoked. (Consequently, it is also set while an item callback is invoked, if the user selected an item.)

- (send *a-popup-menu* get-popup-target) ⇒ canvas<%> or editor<%> object or #f

3.37   `radio-box%`

Implements: `control<%>`

A `radio-box%` control allows the user to select one of number of mutually exclusive items. The items are displayed as a vertical column or horizontal row of labelled **radio buttons**. Unlike a `list-control<%>`, the set of items in a `radio-box%` cannot be changed dynamically.

Whenever the user changes the selected radio button, the radio box's callback procedure is invoked. A callback procedure is provided as an initialization argument when each radio box is created.

- (make-object radio-box% *label choices parent callback style*) ⇒ radio-box% object
    label : string or #f
    choices : list of strings or bitmap% objects
    parent : frame%, dialog%, panel%, or pane% object
    callback : procedure of two arguments: a radio-box% object and a control-event% object
    style = '(vertical) : list of symbols in '(horizontal vertical)

  Creates a radio button set with string or bitmap labels. The *choices* list specifies the radio button labels; the list of choices must be homogenous, either all strings or all bitmaps.

  If an ampersand ("&") occurs in *label*, it is specially parsed; under Windows and X, the character following an ampersand is underlined in the displayed control to indicate a keyboard mnemonic. (Under MacOS, mnemonic underlines are not shown.) The underlined mnemonic character must be a letter or a digit. The user can move the keyboard focus to the radio box by typing the mnemonic when the control's top-level-window contains the keyboard focus. The user must also hold down the Meta or Alt key if the keyboard focus is currently in a control that handles normal alphanumeric input. The ampersand itself is removed from *label* before it is displayed for the control; a double-ampersand in *label* is converted to a single ampersand (with no mnemonic underlining). Mnemonic keyboard events are handled by `on-traverse-char` (but not under MacOS).

  Each string in *choices* can also contain an ampersand, which creates a mnemonic for clicking the corresponding radio button. As for *label*, a double ampersand is converted to a single ampersand.

  If *choices* is a list of bitmaps, then the bitmaps must be valid (see `ok?` in `bitmap%`) and not installed in a `bitmap-dc%` object; otherwise, an `exn:application:mismatch` exception is raised.

  If *label* is a string, it is used as the label for the radio box. Otherwise, the radio box does not display its label.

  The *callback* procedure is called (with the event type '`radio-box`) when the user changes the radio button selection.

  The *style* argument must include either '`vertical` for a collection of radio buttons vertically arranged, or '`horizontal` for a horizontal arrangement.

`enable`

Enables or disables a window so that input events are ignored. (Input events include mouse events, keyboard events, and close-box clicks, but not focus or update events.) When a window is disabled, input events to its children are also ignored.

The enable state of a window can be changed by enabling a parent window, and such changes do not go through this method; use `on-superwindow-enable` to monitor enable state changes.

- (send *a-radio-box* enable *enable?*) ⇒ void
    enable? : boolean

  If *enable?* is #f, the entire radio box is disabled, otherwise it is enabled.

- (send *a-radio-box* enable *n enable?*) ⇒ void
    n : exact non-negative integer
    enable? : boolean

If *enable?* is `#f`, the nth radio button is disabled, otherwise it is enabled (assuming the entire radio box is enabled). Radio buttons are numbered from `0`. If n is equal to or larger than the number of radio buttons in the radio box, an `exn:application:mismatch` exception is raised.

get-item-label

Gets the label of a radio button by position. Radio buttons are numbered from `0`.

- (send *a-radio-box* get-item-label n) $\Rightarrow$ string
    n : exact non-negative integer

    If n is equal to or larger than the number of radio buttons in the radio box, an `exn:application:mismatch` exception is raised.

get-item-plain-label

Like `get-item-label`, except that the label must be a string and ampersands in the label are removed.

- (send *a-radio-box* get-item-plain-label n) $\Rightarrow$ string
    n : exact non-negative integer

    If n is equal to or larger than the number of radio buttons in the radio box, an `exn:application:mismatch` exception is raised.

get-number

Returns the number of radio buttons in the radio box.

- (send *a-radio-box* get-number) $\Rightarrow$ exact non-negative integer

get-selection

Gets the position of the selected radio button. Radio buttons are numbered from `0`.

- (send *a-radio-box* get-selection) $\Rightarrow$ exact non-negative integer

is-enabled?

Returns `#t` if the window is enabled when all of its ancestors are enabled, `#f` otherwise.

- (send *a-radio-box* is-enabled?) $\Rightarrow$ boolean

    Returns `#f` if the entire radio box is disabled, `#t` otherwise.

- (send *a-radio-box* is-enabled? n) $\Rightarrow$ boolean
    n : exact non-negative integer

    Returns `#f` if nth radio button is disabled (independet of disabling the entire radio box), `#t` otherwse. Radio buttons are numbered from `0`. If n is equal to or larger than the number of radio buttons in the radio box, an `exn:application:mismatch` exception is raised.

`set-selection`

Sets the selected radio button by position. (The control's callback procedure is *not* invoked.) Radio buttons are numbered from `0`.

A radio box's selection can be changed by the user clicking the control, and such changes do not go through this method; use the control callback procedure (provided as an initialization argument) to monitor selection changes.

- (send *a-radio-box* `set-selection` *n*) ⇒ void
    n : exact non-negative integer

  If *n* is equal to or larger than the number of radio buttons in the radio box, an `exn:application:mismatch` exception is raised.

3.38   `scroll-event%`

Superclass: `event%`

A `scroll-event%` object contains information about a scroll event. An instance of `scroll-event%` is always provided to `on-scroll`.

See `get-event-type` for a list of the scroll event types.

- (make-object `scroll-event%`) ⇒ `scroll-event%` object

`get-direction`

Gets the identity of the scrollbar that was modified by the event, either the horizontal scrollbar or the vertical scrollbar, as `'horizontal` or `'vertical`, respectively. See also `set-direction`.

- (send *a-scroll-event* `get-direction`) ⇒ symbol in `'(horizontal vertical)`

`get-event-type`

Returns the type of the event, one of the following:

- `'top` — user clicked a scroll-to-top button
- `'bottom` — user clicked a scroll-to-bottom button
- `'line-up` — user clicked an arrow to scroll up or left one step
- `'line-down` — user clicked an arrow to scroll down or right one step
- `'page-up` — user clicked an arrow to scroll up or left one page
- `'page-down` — user clicked an arrow to scroll down or right one page
- `'thumb` — user dragged the scroll position indicator

- (send *a-scroll-event* `get-event-type`) ⇒ symbol in `'(top bottom line-up line-down page-up page-down thumb)`

`get-position`

Returns the position of the scrollbar after the action triggering the event. See also `set-position`.

- (send *a-scroll-event* `get-position`) ⇒ exact integer in [0, 10000]

`set-direction`

Sets the identity of the scrollbar that was modified by the event, either the horizontal scrollbar or the vertical scrollbar, as `'horizontal` or `'vertical`, respectively. See also `get-direction`.

- (send *a-scroll-event* `set-direction` *direction*) ⇒ void
    direction : symbol in `'(horizontal vertical)`

`set-event-type`

Sets the type of the event. See `get-event-type` for information about each event type.

- (send *a-scroll-event* `set-event-type` *type*) ⇒ void
    type : symbol in `'(top bottom line-up line-down page-up page-down thumb)`

`set-position`

Records the position of the scrollbar after the action triggering the event. (The scrollbar itself is unaffected). See also `get-position`.

- (send *a-scroll-event* `set-position` *position*) ⇒ void
    position : exact integer in [0, 10000]

3.39 `selectable-menu-item<%>`

Extends: `labelled-menu-item<%>`

A `selectable-menu-item<%>` object is a `labelled-menu-item<%>` that the user can select. It may also have a keyboard shortcut; the shortcut is displayed in the menu, and the default `on-subwindow-char` method in the menu's frame dispatches to the menu item when the shortcut key combination is pressed.

`command`

Invoke's the menu item's callback procedure, which is supplied when an instance of `menu-item%` or `checkable-menu-item%` is created.

- (send *a-selectable-menu-item* `command` *event*) ⇒ void
    event : `control-event%` object

`get-shortcut`

Gets the keyboard shortcut character for the menu item. Under MacOS, this character is always prefixed with the command modifier. Under Windows, the character is prefixed with the control modifier. Under X,

the modifier depends on the shortcut prefix returned by `get-x-shortcut-prefix`.

If the menu item has no shortcut, `#f` is returned.

The shortcut part of a menu item name is not included in the label returned by `get-label`.

  - (send *a-selectable-menu-item* `get-shortcut`) ⇒ character or `#f`

`get-x-shortcut-prefix`

Returns a symbol that indicates the keyboard prefix used for the menu item's keyboard shortcut. The possible values are the following:

  - `'meta`

  - `'alt`

  - `'ctl`

  - `'ctl-m`

See `get-shortcut` for more information.

  - (send *a-selectable-menu-item* `get-x-shortcut-prefix`) ⇒ symbol in `'(meta alt ctl ctl-m)`

`set-shortcut`

Sets the keyboard shortcut character for the menu item. See `get-shortcut` for more information.

If the shortcut character is set to `#f`, then menu item has no keyboard shortcut.

  - (send *a-selectable-menu-item* `set-shortcut` *shortcut*) ⇒ void
      shortcut : character or `#f`

`set-x-shortcut-prefix`

Sets a symbol that indicates the keyboard prefix used for the menu item's keyboard shortcut.

See `get-x-shortcut-prefix` for more information.

  - (send *a-selectable-menu-item* `set-x-shortcut-prefix` *prefix*) ⇒ void
      prefix : symbol in `'(meta alt ctl ctl-m)`

3.40   `separator-menu-item%`

Implements: `menu-item<%>`

A separator is an unselectable line in a menu. Its parent must be a `menu%` or `popup-menu%`.

72

- (make-object separator-menu-item% *parent*) ⇒ separator-menu-item% object
    parent : menu% or popup-menu% object

Creates a new separator in the menu.

3.41 `slider%`

Implements: `control<%>`

A `slider` object is a panel item with a handle that the user can drag to change the control's value. Each slider has a fixed minumum and maximum value.

Whenever the user changes the value of a slider, its callback procedure is invoked. A callback procedure is provided as an initialization argument when each slider is created.

- (make-object slider% *label min-value max-value parent callback init-value style*) ⇒ slider% object
    label : string or #f
    min-value : exact integer in [-10000, 10000]
    max-value : exact integer in [-10000, 10000]
    parent : frame%, dialog%, panel%, or pane% object
    callback : procedure of two arguments: a slider% object and a control-event% object
    init-value = *min-value* : exact integer in [-10000, 10000]
    style = '(horizontal) : list of symbols in '(horizontal vertical plain)

If *label* is a string, it is used as the label for the slider. Otherwise, the slider does not display its label.

If an ampersand ("&") occurs in *label*, it is specially parsed; under Windows and X, the character following an ampersand is underlined in the displayed control to indicate a keyboard mnemonic. (Under MacOS, mnemonic underlines are not shown.) The underlined mnemonic character must be a letter or a digit. The user can move the keyboard focus to the slider by typing the mnemonic when the control's top-level-window contains the keyboard focus. The user must also hold down the Meta or Alt key if the keyboard focus is currently in a control that handles normal alphanumeric input. The ampersand itself is removed from *label* before it is displayed for the control; a double-ampersand in *label* is converted to a single ampersand (with no mnemonic underlining). Mnemonic keyboard events are handled by `on-traverse-char` (but not under MacOS).

The *min-value* and *max-value* arguments specify the range of the slider, inclusive. The *init-value* argument optionally specifies the slider's initial value. If the sequence [*min-value*, *initial-value*, *maximum-value*] is not increasing, an `exn:application:mismatch` exception is raised.

The *callback* procedure is called (with the event type 'slider) when the user changes the slider's value.

The *style* argument must include either 'vertical for a vertical slider, or 'horizontal for a horizontal slider. If *style* includes 'plain, the slider does not display numbers for its range and current value to the user.

`get-value`

Gets the current slider value.

- (send *a-slider* get-value) ⇒ exact integer in [-10000, 10000]

`set-value`

Sets the value (and displayed position) of the slider. (The control's callback procedure is *not* invoked.)

A slider's value can be changed by the user clicking the control, and such changes do not go through this method; use the control callback procedure (provided as an initialization argument) to monitor value changes.

- (send *a-slider* `set-value` *value*) $\Rightarrow$ void
    value : exact integer in [-10000, 10000]

   If *value* is outside the slider's minimum and maximum range, an `exn:application:mismatch` exception is raised.

3.42 subarea<%>

Extends: `area<%>`

A `subarea<%>` is a containee `area<%>`.

`horiz-margin`

Gets or sets the area's horizontal margin, which is added both to the right and left, for geometry management. See §2.2 for more information.

- (send *a-subarea* `horiz-margin`) $\Rightarrow$ exact integer in [0, 1000]

   Returns the current horizontal margin.

- (send *a-subarea* `horiz-margin` *margin*) $\Rightarrow$ void
    margin : exact integer in [0, 1000]

   Sets the horizontal margin.

`vert-margin`

Gets or sets the area's vertical margin, which is added both to the right and left, for geometry management. See §2.2 for more information.

- (send *a-subarea* `vert-margin`) $\Rightarrow$ exact integer in [0, 1000]

   Returns the current vertical margin.

- (send *a-subarea* `vert-margin` *margin*) $\Rightarrow$ void
    margin : exact integer in [0, 1000]

   Sets the vertical margin.

3.43 subwindow<%>

Extends: `subarea<%>`

Extends: `window<%>`

A `subwindow<%>` is a containee window.

3.44 `text-field%`

Implements: `control<%>`

A `text-field%` object is an editable text field with an optional label displayed in front of it. There are two text field styles:

- A single line of text is visible, and a special control event is generated when the user presses Enter (when the text field has the focus) and the event is not handled by the text field's frame or dialog (see `on-traverse-char` in `top-level-window<%>`).

- Multiple lines of text are visible, and Enter is not handled specially.

Whenever the user changes the content of a text field, its callback procedure is invoked. A callback procedure is provided as an initialization argument when each text field is created.

The text field is implemented using a `text%` editor (with an inaccessible display). Thus, whereas `text-field%` provides only `get-value` and `set-value` to manipulate the text in a text field, the `get-editor` returns the field's editor, which provides a vast collection of methods for more sophisticated operations on the text.

The keymap for the text field's editor is initialized by calling the current keymap initializer procedure, which is determined by the `current-text-keymap-initializer` parameter.

- (make-object `text-field%` *label parent callback init-value style*) ⇒ `text-field%` object
    label : string or `#f`
    parent : `frame%`, `dialog%`, `panel%`, or `pane%` object
    callback : procedure of two arguments: a `text-field%` object and a `control-event%` object
    init-value = `""` : string
    style = `'(single)` : list of symbols in `'(single multiple hscroll)`

If *label* is not `#f`, it is used as the text field label. Otherwise, the text field will does not display its label.

If an ampersand ("&") occurs in *label*, it is specially parsed; under Windows and X, the character following an ampersand is underlined in the displayed control to indicate a keyboard mnemonic. (Under MacOS, mnemonic underlines are not shown.) The underlined mnemonic character must be a letter or a digit. The user can move the keyboard focus to the text field by typing the mnemonic when the control's top-level-window contains the keyboard focus. The user must also hold down the Meta or Alt key if the keyboard focus is currently in a control that handles normal alphanumeric input. The ampersand itself is removed from *label* before it is displayed for the control; a double-ampersand in *label* is converted to a single ampersand (with no mnemonic underlining). Mnemonic keyboard events are handled by `on-traverse-char` (but not under MacOS).

The *callback* procedure is called when the user changes the text in the text field or presses the Enter key (and Enter is not handled by the text field's frame or dialog; see `on-traverse-char` in `top-level-window<%>`). If the user presses Enter, the type of event passed to the callback is `'text-field-enter`, otherwise it is `'text-field`.

If `init-value` is not `""`, the minimum width of the text item is made wide enough to show `init-value`. Otherwise, a built-in default width is selected. For a text field in single-line mode, the minimum height is set to show one line and only the control's width is stretchable. For a multiple-line text field, the minimum height shows three lines of text and is stretchable in both directions.

The style must contain exactly one of `'single` or `'multiple`; the former specifies a single-line field and the latter specifies a multiple-line field. The `'hscroll` style applies only to multiple-line fields; when `'hscroll` is specified, the field has a horizontal scrollbar and autowrapping is disabled; otherwise, the field has no horizontal scrollbar and autowrapping is enabled. A multiple-line text field always has a vertical scrollbar.

`get-editor`

Returns the editor used to implement the text field.

For a text field, the most useful methods of a `text%` object are the following:

- (send *a-text* `get-text`) returns the current text of the editor.

- (send *a-text* `erase`) deletes all text from the editor.

- (send *a-text* `insert` *string*) inserts *string* into the editor at the current caret position.

  - (send *a-text-field* `get-editor`) ⇒ `text%` object

`get-value`

Returns the text currently in the text field.

  - (send *a-text-field* `get-value`) ⇒ string

`set-value`

Sets the text currently in the text field. (The control's callback procedure is *not* invoked.)

A text field's value can be changed by the user typing into the control, and such changes do not go through this method; use the control callback procedure (provided as an initialization argument) to monitor value changes.

  - (send *a-text-field* `set-value` *val*) ⇒ void
    val : string

3.45 `timer%`

A `timer%` object encapsulates an event-based alarm. To use a timer, derive a new class and override the `notify` method to perform the alarm-based action. Start a timer with `start` and stop it with `stop`.

Timers have a relatively high priority in the event queue. Thus, if the timer delay is set low enough, repeated notification for a timer can preempt user activities (which might be directed at stopping the timer). For timers with relatively short delays, call `yield` within the `notify` procedure to allow guranteed event processing.

See §2.4 for more information about event priorities.

  - (make-object `timer%`) ⇒ `timer%` object
    Creates an idle timer (i.e., the timer is not started, so `notify` will not be called).

interval

Returns the number of millseconds between each timer expiration (when the timer is running).

- (send *a-timer* `interval`) ⇒ exact integer in [0, 1000000000]

notify

Called (on an event boundary) when the timer's alarm expires.

- (send *a-timer* `notify`) ⇒ void

start

Starts (or restarts) the timer. If the timer is already running, its alarm time is not changed.

- (send *a-timer* `start` *m* *just-once?*) ⇒ void
  m : exact integer in [0, 1000000000]
  just-once? = `#f` : boolean

  The timer's alarm expires after `m` milliseconds, at which point `notify` is called (on an event boundary). If *just-once?* is `#f`, the timer expires *every* `m` milliseconds until the timer is explictly stopped;[1] otherwise, the timer expires only once.

stop

Stops the timer. A stopped timer never calls `notify`. If the timer has expired but the call to `notify` has not yet been dispatched, the call is removed from the event queue.

- (send *a-timer* `stop`) ⇒ void

3.46   `top-level-window<%>`

Extends: `area-container-window<%>`

A top-level window is either a `frame%` or `dialog%` object.

can-close?

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

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

- (send *a-top-level-window* `can-close?`) ⇒ boolean

---

[1]Actually, the timer expires `m` milliseconds after `notify` returns each time

`can-exit?`

Called for each visible top-level window when the operating system requests that all programs shut down; for each frame that returns a true value, the frame's `on-exit` method is called.

- (send *a-top-level-window* `can-exit?`) ⇒ boolean

  Calls `can-close?` and returns the result.

`center`

Centers the window on the screen if it has no parent. If it has a parent, the window is centered with respect to its parent's location.

- (send *a-top-level-window* `center` *direction*) ⇒ void
    direction = 'both : symbol in '(horizontal vertical both)

  If *direction* is '`horizontal`, the window is centered horizontally. If *direction* is '`vertical`, the window is centered vertically. If *direction* is '`both`, the window is centered in both directions.

`get-edit-target-object`

Like `get-edit-target-window`, but if an editor canvas had the focus and it also displays an editor, the editor is returned instead of the canvas. Further, if the editor's focus is delegated to an embedded editor, the embedded editor is returned.

See also `get-focus-object`.

- (send *a-top-level-window* `get-edit-target-object`) ⇒ window<%> or editor<%> object or #f

`get-edit-target-window`

Returns the window that most recently had the keyboard focus, either the top-level window or one of its currently-shown children. If neither the window nor any of its currectly-shown children has even owned the keyboard focus, #f is returned.

See also `get-focus-window` and `get-edit-target-object`.

- (send *a-top-level-window* `get-edit-target-window`) ⇒ window<%> object or #f

`get-eventspace`

Returns the window's eventspace.

- (send *a-top-level-window* `get-eventspace`) ⇒ eventspace

`get-focus-object`

Like `get-focus-window`, but if an editor canvas has the focus and it also displays an editor, the editor is returned instead of the canvas. Further, if the editor's focus is delegated to an embedded editor, the embedded editor is returned.

See also `get-edit-target-object`.

  - (send *a-top-level-window* `get-focus-object`) ⇒ `window<%>` or `editor<%>` object or `#f`

get-focus-window

Returns the window that has the keyboard focus, either the top-level window or one of its children. If neither the window nor any of its children has the focus, `#f` is returned.

See also `get-edit-target-window` and `get-focus-object`.

  - (send *a-top-level-window* `get-focus-window`) ⇒ `window<%>` object or `#f`

move

Moves the window to the given position on the screen.

A window's position can be changed by the user dragging the window, and such changes do not go through this method; use `on-move` to monitor position changes.

  - (send *a-top-level-window* `move` *x y*) ⇒ void
      x : exact integer in [-10000, 10000]
      y : exact integer in [-10000, 10000]

on-activate

Called when a window is **activated** or **deactivated**. A top-level window is activated when the keyboard focus moves from outside the window to the window or one of its children. It is deactivated when the focus moves back out of the window.

The method's argument is `#t` when the window is activated, `#f` when it is deactivated.

  - (send *a-top-level-window* `on-activate` *active?*) ⇒ void
      active? : boolean

on-close

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

See also `can-close?`.

  - (send *a-top-level-window* `on-close`) ⇒ void

on-exit

Called for each visible top-level window when the operating system requests that all programs shut down. For each top-level window, this method is called only if the frame's `can-exit?` method returns true.

  - (send *a-top-level-window* `on-exit`) ⇒ void

    Calls `on-close` and then `show` to hide the window.

`on-message`

A generic message method, usually called by `send-message-to-window`.

If the method is invoked by `send-message-to-window`, then it is invoked in the thread where `send-message-to-window` was called (which is possibly *not* the handler thread of the window's eventspace).

- (send *a-top-level-window* `on-message` *message*) ⇒ value
    message : value

    Returns void.

`on-system-menu-char`

Checks whether the given event pops open the system menu in the top-left corner of the window (Windows only). If the window's system menu is opened, `#t` is returned, otherwise `#f` is returned.

- (send *a-top-level-window* `on-system-menu-char` *event*) ⇒ boolean
    event : `key-event%` object

`on-traverse-char`

Attempts to handle the given keyoard event as a navigation event, such as a Tab key event that moves the keyboard focus. If the event is handled, `#t` is returned, otherwise `#f` is returned.

- (send *a-top-level-window* `on-traverse-char` *event*) ⇒ boolean
    event : `key-event%` object

    The following rules determine, in order, whether and how *event* is handled:

    – If the window that currently owns the focus specifically handles the event, then `#f` is returned. The following describes window types and the keyboard events they specifically handle:
        * `editor-canvas%` — tab-exit is disabled (see `allow-tab-exit`): all keyboard events, except alphanumeric key events when the Meta (X) or Alt (Windows) key is pressed; when tab-exit is enabled: all keyboard events except Tab, Enter, Escape, and alphanumeric Meta/Alt events.
        * `canvas%` — when tab-focus is disabled (see `accept-tab-focus`): all keyboard events, except alphanumeric key events when the Meta (X) or Alt (Windows) key is pressed; when tab-focus is enabled: no key events
        * `text-field%`, `'single` style — arrow key events and alphanumeric key events when the Meta (X) or Alt (Windows) key is not pressed (and all alphanumeric events under MacOS)
        * `text-field%`, `'multiple` style — all keyboard events, except alphanumeric key events when the Meta (X) or Alt (Windows) key is pressed
        * `choice%` — arrow key events and alphanumeric key events when the Meta (X) or Alt (Windows) key is not pressed
        * `list-box%` — arrow key events and alphanumeric key events when the Meta (X) or Alt (Windows) key is not pressed
    – If *event* is a Tab or arrow key event, the keyboard focus is moved within the window and `#t` is returned. Across platforms, the types of windows that accept the keyboard focus via navigation may vary, but `text-field%` windows always accept the focus, and `message%`, `gauge%`, and `panel%` windows never accept the focus.
    – If *event* is an Space key event and the window that currently owns the focus is a `button%`, `check-box%`, or `radio-box%` object, the event is handled in the same way as a click on the control and `#t` is returned.

– If *event* is an Enter key event and the current top-level window contains a border button, the button's callback is invoked and `#t` is returned. (The `'border` style for a `button%` object indicates to the user that pressing Enter is the same as clicking the button.) If the window does not contain a border button, `#t` is returned if the window with the current focus is not a text field or editor canvas.

– In a dialog, if *event* is an Escape key event, the event is handled the same as a click on the dialog's close box (i.e., the dialog's `can-close?` and `on-close` methods are called, and the dialog is hidden) and `#t` is returned.

– If *event* is an alphanumeric key event and the current top-level window contains a control with a mnemonic matching the key (which is installed via a label that contains "&"; see `get-label` for more information), then the keyboard focus is moved to the matching control. Furthermore, if the matching control is a `button%`, `check-box%`, or `radio-box%` button, the keyboard event is handled in the same way as a click on the control.

– Otherwise, `#f` is returned.

`resize`

Sets the size of the window (in pixels), but only if the given size is larger than the window's minimum size.

A window's size can be changed by the user, and such changes do not go through this method; use `on-size` to monitor size changes.

- (send *a-top-level-window* `resize` *width* *height*) ⇒ void
    width : exact integer in [0, 10000]
    height : exact integer in [0, 10000]

`show`

Shows or hides a window.

The visibility of a window can be changed by the user clicking the window's close box, for example, , and such changes do not go through this method; use `on-superwindow-show` or `on-close` to monitor visibility changes.

- (send *a-top-level-window* `show` *show*) ⇒ void
    show : boolean

If *show?* is `#f`, the window is hidden. Otherwise, the window is shown.

If the window is already shown, it is moved front of other top-level windows. If the window is iconized (frames only), it is deiconized.

3.47 `vertical-pane%`

Superclass: `pane%`

A vertical pane arranges its subwindows in a single column. See also `pane%`.

- (make-object `vertical-pane%` *parent*) ⇒ `vertical-pane%` object
    parent : `frame%`, `dialog%`, `panel%`, or `pane%` object

3.48   `vertical-panel%`

Superclass: `panel%`

A vertical panel arranges its subwindows in a single column. See also `panel%`.

- (make-object vertical-panel% *parent style*) ⇒ `vertical-panel%` object
    parent : `frame%`, `dialog%`, `panel%`, or `pane%` object
    style = `null` : list of symbols in `'(border)`

    If the `'border` style is specified, the window is created with a thin border (only in this case, the client size of the panel may be less than its total size).

3.49   `window<%>`

Extends: `area<%>`

A `window<%>` object is an `area<%>` with a graphical representation that can respond to events.

`accept-drop-files`

Enables or disables drag-and-drop dropping for the window (on platforms that support drag-and-drop), or gets the enable state. Dropping is initially disabled. See also `on-drop-file` .

- (send *a-window* accept-drop-files) ⇒ boolean

    Returns `#t` if file-dropping is enabled, `#f` otherwise.

- (send *a-window* accept-drop-files *accept-files?*) ⇒ void
    accept-files? : boolean

    Enables file-dropping if *accept-files?* is true, disables file-dropping otherwise.

`client->screen`

Converts local window coordinates to screen coordinates.

- (send *a-window* client->screen *x y*) ⇒ two exact integers in [-10000, 10000]
    x : exact integer in [-10000, 10000]
    y : exact integer in [-10000, 10000]

`enable`

Enables or disables a window so that input events are ignored. (Input events include mouse events, keyboard events, and close-box clicks, but not focus or update events.) When a window is disabled, input events to its children are also ignored.

The enable state of a window can be changed by enabling a parent window, and such changes do not go through this method; use `on-superwindow-enable` to monitor enable state changes.

- (send *a-window* enable *enable?*) ⇒ void
    enable? : boolean

If *enable?* is true, the window is enabled, otherwise it is disabled.

focus

Moves the keyboard focus to the window, relative to its top-level window. If the focus is in the window's top-level window, then the focus is immediately moved to this window. Otherwise, the focus is not immediately moved, but when the window's top-level window gets the keyboard focus, it is delegated to this window.

See also `on-focus`.

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

The current keyboard focus window can be changed by the user, and such changes do not go through this method; use `on-focus` to monitor focus changes.

- (send *a-window* focus) ⇒ void

get-client-size

This gets the interior size of the window. For a container, the interior size is the size available for placing subwindows (including the border margin). For a canvas, this is the visible drawing area.

The client size is returned as two values: width and height (in pixels).

- (send *a-window* get-client-size) ⇒ two exact integers in [0, 10000]

get-cursor

Returns the window's cursor, or `#f` if this window's cursor defaults to the parent's cursor. See `set-cursor` for more information.

- (send *a-window* get-cursor) ⇒ cursor% object or #f

get-height

Returns the window's total height (in pixels).

See also `reflow-container`.

- (send *a-window* get-height) ⇒ exact integer in [0, 10000]

get-label

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

The label string may contain ampersands ("&"), which serve as keyboard navigation annotations for controls under Windows and X. The ampersands are not part of the displayed label of a control; instead, ampersands

are removed in the displayed label (under all platforms), and any character preceeding an ampersand is underlined (Windows and X) indicating that the character is a mnemonic for the control. Double ampersands are converted into a single ampersand (with no displayed underline). See also `on-traverse-char`.

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

- (send *a-window* `get-label`) ⇒ string, `bitmap%` object, or `#f`

`get-plain-label`

Like `get-label`, except that ampersands in the label are removed. If the window's label is not a string, `#f` is returned.

- (send *a-window* `get-plain-label`) ⇒ string or `#f`

`get-size`

This gets the size of the entire window in pixels, not counting horizontal and vertical margins. (Under X, this size does not include a title bar or borders for a frame/dialog.) See also `get-client-size`.

The geometry is returned as two values: width and height (in pixels).

- (send *a-window* `get-size`) ⇒ two exact integers in [0, 10000]

`get-width`

Returns the window's total width (in pixels).

See also `reflow-container`.

- (send *a-window* `get-width`) ⇒ exact integer in [0, 10000]

`get-x`

Returns the position of the window's left edge in its parent's coordinate system.

See also `reflow-container`.

- (send *a-window* `get-x`) ⇒ exact integer in [-10000, 10000]

`get-y`

Returns the position of the window's top edge in its parent's coordinate system.

See also `reflow-container`.

- (send *a-window* `get-y`) ⇒ exact integer in [-10000, 10000]

`has-focus?`

Indicates whether the window currently has the keyboard focus. See also `on-focus`.

- (send *a-window* `has-focus?`) $\Rightarrow$ boolean

`is-enabled?`

Returns `#t` if the window is enabled when all of its ancestors are enabled, `#f` otherwise.

- (send *a-window* `is-enabled?`) $\Rightarrow$ boolean

`is-shown?`

Indicates whether the window is currently shown or not (when all of its ancestors are also shown).

The result is `#t` if this window is shown when its ancestors are shown, or `#f` if this window remains hidden when its ancestors are shown.

- (send *a-window* `is-shown?`) $\Rightarrow$ boolean

`on-drop-file`

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

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

- (send *a-window* `on-drop-file` *pathname*) $\Rightarrow$ void
    pathname : string

`on-focus`

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

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

- (send *a-window* `on-focus` *on?*) $\Rightarrow$ void
    on? : boolean
  Does nothing.

`on-move`

Called when the window is moved. (For windows that are not top-level windows, "moved" means moved relative to the parent's top-left corner.) The new position is provided to the method.

- (send *a-window* `on-move` *x* *y*) $\Rightarrow$ void
    x : exact integer in [-10000, 10000]
    y : exact integer in [-10000, 10000]

Does nothing.

on-size

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

- (send *a-window* on-size *width height*) $\Rightarrow$ void
    width : exact integer in [0, 10000]
    height : exact integer in [0, 10000]

Does nothing.

on-subwindow-char

Called when this window or a child window receives a keyboard event. The `on-subwindow-char` method of the receiver's top-level window is called first (see `get-top-level-window`); if the return value is `#f`, then the `on-subwindow-char` method is called for the next child in the path to the receiver, and so on. Finally, if the receiver's `on-subwindow-char` method returns `#f`, the event is passed on to the receiver's normal key-handling mechanism.

BEWARE: The default `on-subwindow-char` in `frame%` and `on-subwindow-char` in `dialog%` methods consume certain keyboard events (e.g., arrow keys, Enter) used for navigating within the window. Because the top-level window gets the first chance to handle the keyboard event, some events never reach the "receiver" child unless the default frame or dialog method is overridden.

- (send *a-window* on-subwindow-char *receiver event*) $\Rightarrow$ boolean
    receiver : `window<%>` object
    event : `key-event%` object

The *event* argument is the event that was generated for the *receiver* window. Returns `#f`.

on-subwindow-event

Called when this window or a child window receives a mouse event. The `on-subwindow-event` method of the receiver's top-level window is called first (see `get-top-level-window`); if the return value is `#f`, the `on-subwindow-event` method is called for the next child in the path to the receiver, and so on. Finally, if the receiver's `on-subwindow-event` method returns `#f`, the event is passed on to the receiver's normal mouse-handling mechanism.

- (send *a-window* on-subwindow-event *receiver event*) $\Rightarrow$ boolean
    receiver : `window<%>` object
    event : `mouse-event%` object

The *event* argument is the event that was generated for the *receiver* window. Returns `#f`.

on-superwindow-enable

Called via the event queue whenever the enable state of a window has changed, either through a call to the window's `enable` method, or through the enabling/disabling of one of the window's ancestors. The method's argument indicates whether the window is now enabled or not.

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

If the enable state of a window's ancestor changes while the window is not active (e.g., because it was removed with `delete-child`), then no enable events are queued for the inactive window. But if the window is later re-activated into an enable state that is different from the window's state when it was de-activated, then an enable event is immediately queued.

- (send *a-window* `on-superwindow-enable` *enabled?*) ⇒ void
    enabled? : boolean

on-superwindow-show

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

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

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

refresh

Enqueues an event to repaint the window.

- (send *a-window* `refresh`) ⇒ void

screen->client

Converts global coordinates to widnow local coordinates.

- (send *a-window* `screen->client` *x* *y*) ⇒ two exact integers in [-10000, 10000]
    x : exact integer in [-10000, 10000]
    y : exact integer in [-10000, 10000]

set-cursor

Sets the window's cursor. Providing `#f` instead of a cursor value removes the window's cursor.

If a window does not have a cursor, it uses the cursor of its parent. Frames and dialogs start with the standard arrow cursor, and text fields start with an I-beam cursor. All other windows are created without a cursor.

- (send *a-window* `set-cursor` *cursor*) ⇒ void
    cursor : `cursor%` object or `#f`

`set-label`

Sets a window's label. The window's natural minimum size might be different after the label is changed, but the window's mininum size is not recomputed.

See `get-label` for more information.

- (send *a-window* `set-label` *l*) ⇒ void
  l : string or #f

  If *l* is #f, the window's label is removed.

`show`

Shows or hides a window.

The visibility of a window can be changed by the user clicking the window's close box, for example, , and such changes do not go through this method; use `on-superwindow-show` or `on-close` to monitor visibility changes.

- (send *a-window* `show` *show?*) ⇒ void
  show? : boolean

  If *show?* is #f, the window is hidden. Otherwise, the window is shown.

4. Windowing Procedures

4.1 Dialogs

These functions get input from the user and/or display messages.

`get-choices-from-user`

- (`get-choices-from-user` *title message choices parent init-choices style*) ⇒ list of exact non-negative integers or `#f`
  title : string
  message : string or `#f`
  choices : list of strings
  parent = `#f` : `frame%` or `dialog%` object or `#f`
  init-choices = `null` : list of exact non-negative integers
  style = `'(single)` : list of symbols in `'(single multiple extended)`

  Gets a list box selection from the user via a modal dialog, using *parent* as the parent window if it is specified. The dialog's title is *title*. The dialog's list box is labelled with *message* and initialized by selecting the items in *init-choices*.

  The style must contain exactly one of `'single`, `'multiple`, or `'extended`. The styles have the same meaning as for creating a `list-box%` object. (For the single-selection style, only the last selection in *init-choices* matters.)

  The result is `#f` if the user cancels the dialog, the list of selections otherwise.

`get-color-from-user`

- (`get-color-from-user` *message parent init-color style*) ⇒ `color%` object or `#f`
  message = `#f` : string or `#f`
  parent = `#f` : `frame%` or `dialog%` object or `#f`
  init-color = `#f` : `color%` object or `#f`
  style = `null` : an empty list of symbols

  Lets the user select a color though the platform-specific (modal) dialog, using *parent* as the parent window if it is specified. The *message* string is displayed as a prompt in the dialog if possible. If *init-color* is provided, the dialog is initialized to the given color.

  The result is `#f` if the user cancels the dialog, the selected color otherwise.

  The *style* argument is provided for future extensions. Currently, *style* must be the empty list.

`get-file`

- (`get-file` *message parent directory filename extension style*) ⇒ string or `#f`
  message = `#f` : string or `#f`
  parent = `#f` : `frame%` or `dialog%` object or `#f`

    directory = `#f` : string or `#f`
    filename = `#f` : string or `#f`
    extension = `#f` : string or `#f`
    style = `null` : an empty list of symbols

Obtains a file pathname from the user via the platform-specific standard (modal) dialog, using *parent* as the parent window if it is specified.

The result is `#f` if the user cancels the dialog, the selected pathname otherwise. The returned pathname may or may not exist, although the style of the dialog is directed towards selecting existing files.

If *directory* is not `#f`, it is used as the starting directory for the file selector (otherwise the starting directory is chosen automatically in a platform-specific manner, usually based on the current directory and the user's interactions in previous calls to `get-file` and `put-file`). If *filename* is not `#f`, it is used as the default filename when appropriate.

Under Windows, if *extension* is not `#f`, the returned path will use the extension if the user does not supply one; the *extension* string should not contain a period. The extension is ignored on other platforms.

The *style* argument is provided for future extensions. Currently, *style* must be the empty list.

get-file-list

Like `get-file`, except that the user can select multiple files, and the result is either a list of file paths of `#f`.

- (`get-file-list` *message parent directory filename extension style*) ⇒ list of strings or `#f`
    message = `#f` : string or `#f`
    parent = `#f` : `frame%` or `dialog%` object or `#f`
    directory = `#f` : string or `#f`
    filename = `#f` : string or `#f`
    extension = `#f` : string or `#f`
    style = `null` : an empty list of symbols

get-font-from-user

- (`get-font-from-user` *message parent init-font style*) ⇒ `font%` object or `#f`
    message = `#f` : string or `#f`
    parent = `#f` : `frame%` or `dialog%` object or `#f`
    init-font = `#f` : `font%` object or `#f`
    style = `null` : an empty list of symbols

Lets the user select a font though the platform-specific (modal) dialog, using *parent* as the parent window if it is specified. The *message* string is displayed as a prompt in the dialog if possible. If *init-font* is provided, the dialog is initialized to the given font.

The result is `#f` if the user cancels the dialog, the selected font otherwise.

The *style* argument is provided for future extensions. Currently, *style* must be the empty list.

get-ps-setup-from-user

- (`get-ps-setup-from-user` *message parent init-setup style*) ⇒ `ps-setup%` object or `#f`
    message = `#f` : string or `#f`
    parent = `#f` : `frame%` or `dialog%` object or `#f`
    init-setup = `#f` : `ps-setup%` object or `#f`
    style = `null` : an empty list of symbols

Lets the user select a PostScript configuration though a (modal) dialog, using *parent* as the parent window if it is specified. The *message* string is displayed as a prompt in the dialog. If *init-setup* is provided, the dialog is initialized to the given configuration, otherwise the current configuration from `current-ps-setup` is used.

The result is `#f` if the user cancels the dialog, the selected PostScript configuration otherwise.

The *style* argument is provided for future extensions. Currently, *style* must be the empty list.

get-text-from-user

- (`get-text-from-user` *title* *message* *parent* *init-val* *style*) ⇒ string or `#f`
  title : string
  message : string or `#f`
  parent = `#f` : `frame%` or `dialog%` object or `#f`
  init-val = `""` : string
  style = `null` : an empty list of symbols

Gets a text string from the user via a modal dialog, using *parent* as the parent window if it is specified. The dialog's title is *title*. The dialog's text field is labelled with *message* and initialized to *init-val* (but *init-val* does not determine the size of the dialog).

The result is `#f` if the user cancels the dialog, the user-provided string otherwise.

The *style* argument is provided for future extensions. Currently, *style* must be the empty list.

message-box

- (`message-box` *title* *message* *parent* *style*) ⇒ symbol in '(ok cancel yes no)
  title : string
  message : string
  parent = `#f` : `frame%` or `dialog%` object or `#f`
  style = '(ok) : list of symbols in '(ok ok-cancel yes-no)

Displays a message to the user in a (modal) dialog, using *parent* as the parent window if it is specified. The dialog's title is *title*. The *message* string can be arbitrarily long, and can contain explicit linefeeds or carriage returns for breaking lines.

The style must include exactly one of the following:

- '`ok` — the dialog only has an "Ok" button and always returns '`ok`.
- '`ok-cancel` — the message dialog has "Cancel" and "Ok" buttons. If the user clicks "Cancel", the result is '`cancel`, otherwise the result is '`ok`.
- '`yes-no` — the message dialog has "Yes" and "No" buttons. If the user clicks "Yes", the result is '`yes`, otherwise the result is '`no`.

put-file

- (`put-file` *message* *parent* *directory* *filename* *extension* *style*) ⇒ string or `#f`
  message = `#f` : string or `#f`
  parent = `#f` : `frame%` or `dialog%` object or `#f`
  directory = `#f` : string or `#f`
  filename = `#f` : string or `#f`
  extension = `#f` : string or `#f`
  style = `null` : an empty list of symbols

Obtains a file pathname from the user via the platform-specific standard (modal) dialog, using *parent* as the parent window if it is specified.

The result is `#f` if the user cancels the dialog, the selected pathname otherwise. The returned pathname may or may not exist, although the style of the dialog is directed towards creating a new file.

If *directory* is not `#f`, it is used as the starting directory for the file selector (otherwise the starting directory is chosen automatically in a platform-specific manner, usually based on the current directory and the user's interactions in previous calls to `get-file` and `put-file`). If *filename* is not `#f`, it is used as the default filename when appropriate.

Under Windows, if *extension* is not `#f`, the returned path will use the extension if the user does not supply one; the *extension* string should not contain a period. The extension is ignored on other platforms.

The *style* argument is provided for future extensions. Currently, *style* must be the empty list.

4.2 Eventspaces

`check-for-break`

Inspects the event queue of the current eventspace, seaching for a Shift-Ctl-C (X, Windows) or Cmd-. (MacOS) key combination. Returns `#t` if such an event was found (and the event is dequeued) or `#f` otherwise.

- (check-for-break) ⇒ boolean

`current-eventspace`

This is a parameter (see parameters, §9.4 in *PLT MzScheme: Language Manual*) that obtains or sets the current eventspace.

See section 2.4 (page 13) for more inforamtion about eventspaces.

- (current-eventspace) ⇒ eventspace

  Gets the current eventspace.

- (current-eventspace *e*) ⇒ void
    e : eventspace

  Sets the current eventspace to *e*.

`event-dispatch-handler`

This parameter (see parameters, §9.4 in *PLT MzScheme: Language Manual*) gets or installs the current event dispatch handler. The event dispatch handler is called by an eventspace's handler thread for every queue-based event to be processed in the eventspace. The only argument to the handler is the eventspace in which an event should be dispatched. The event dispatch handler gives the programmer control over the timing of event disptaching, but not the order in which events are dispatched within a single eventspace.

An event dispatch handler must ultimately call the primitive event dispatch handler. If an event dispatch handler returns without calling the primitive handler, then the primitive handler is called directly by the eventspace handler thread.

- (event-dispatch-handler) ⇒ procedure of one argument: an eventspace

  Returns the current handler.

- (event-dispatch-handler *handler*) ⇒ void
    handler : procedure of one argument: an eventspace

  Sets the current handler.

eventspace-shutdown?

Returns #t if the given eventspace has been shut down by its custodian, #f otherwise. Attempting to create a new window, timer, or explicitly queued event in a shut-down eventspace raises the exn:misc exception.

Attempting to use certain methods of windows and timers in a shut-down eventspace also raises the exn:misc exception, but the get-top-level-window in area<%> and get-eventspace in top-level-window<%> methods work even after the area's eventspace is shut down.

- (eventspace-shutdown?   *e*) ⇒ boolean
    e : eventspace

eventspace?

Tests whether a value is an eventspace.

See section 2.4 (page 13) for more inforamtion about eventspaces.

- (eventspace?   *v*) ⇒ boolean
    v : value

  Returns #t if *v* is an eventspace value or #f otherwise.

get-top-level-edit-target-window

Returns the top level window in the current eventspace that is visible and most recently had the keyboard focus (or contains the window that had the keyboard focus), or #f if there is no visible window in the current eventspace.

- (get-top-level-edit-target-window) ⇒ frame% or dialog% object or #f

get-top-level-focus-window

Returns the top level window in the current eventspace that has the keyboard focus (or contains the window with the keyboard focus), or #f if no window in the current eventspace has the focus.

- (get-top-level-focus-window) ⇒ frame% or dialog% object or #f

get-top-level-windows

Returns a list of visible top-level frames and dialogs in the current eventspace.

- (get-top-level-windows) ⇒ list of frame% and dialog% objects

`make-eventspace`

Creates an returns a new eventspace value. The new eventspace is created as a child of the current eventspace. The eventspace is used by making it the current eventspace with the `current-eventspace` parameter.

See section 2.4 (page 13) for more inforamtion about eventspaces.

- (make-eventspace) ⇒ eventspace

`queue-callback`

Installs a procedure to be called via the current eventspace's event queue. The procedure is called once in the same way and under the same restrictions that a callback is invoked to handle a method.

A second (optional) boolean argument indicates whether the callback has a high or low priority in the event queue. See section 2.4 (page 13) for information about the priority of events.

- (queue-callback *callback high-priority?*) ⇒ void
    callback : procedure of no arguments
    high-priority? = #t : boolean

`sleep/yield`

Blocks for at least the specified number of seconds, handling events meanwhile if the current thread is the current eventspace's handler thread (otherwise, `sleep/yield` is equivalent to `sleep`).

- (sleep/yield *secs*) ⇒ void
    secs : non-negative real number

`special-control-key`

Enables or disables special Control key handling (MacOS). When Control is treated as a special key, the system's key-mapper is called without Control for keyboard translations. For some languages, Control key presses must be seen by the system translation, so this mode should be turned off.

- (special-control-key *on?*) ⇒ void
    on? : boolean
  If *on?* is #f, Control is passed to the system translation as normal.

`yield`

Yields control to event dispatching. See §2.4 for details.

A handler procedure invoked by the system during a call to `yield` can itself call `yield`, creating an additional level of nested (but single-threaded) event handling.

See also `sleep/yield` .

- (yield) ⇒ void
    Dispatches an unspecified number of events, but only if the current thread is the current eventspace's handler thread (otherwise, there is no effect).

- (`yield` *wait-symbol*) ⇒ void
  wait-symbol : `'wait`

When called in the handler thread of an eventspace, yields (processing events) until

  – no top-level windows in the eventspace are visible;
  – no timers in the eventspace are running; and
  – no callbacks are queued in the eventspace.

(When called in a non-handler thread, returns immediately.)

- (`yield` *sema*) ⇒ void
  sema : `semaphore`

Blocks on *sema*. If the current thread is the current eventspace's handler thread, events are dispatched until a *sema* wait succeeds on an event boundary. For other threads, calling `yield` with a semaphore is equivalent to calling `semaphore-wait`.

Always use (`yield` *sema*) instead of a busy-wait loop.

4.3 Miscellaneous

`begin-busy-cursor`

- (`begin-busy-cursor`) ⇒ void

Changes the cursor to a watch cursor for all windows in the current eventspace. Use `end-busy-cursor` to revert the cursor back to its previous state. Calls to `begin-busy-cursor` and `end-busy-cursor` can be nested arbitrarily.

The cursor installed by `begin-busy-cursor` overrides any window-specific cursors installed with `set-cursor`.

See also `is-busy?`.

`bell`

- (`bell`) ⇒ void

Rings the system bell.

`end-busy-cursor`

- (`end-busy-cursor`) ⇒ void

See `begin-busy-cursor`.

`find-graphical-system-path`

Finds a platform-specific (and possibly user- or machine-specific) standard filename or directory.

See also `find-system-path`, §11.2.1 in *PLT MzScheme: Language Manual*.

- (`find-graphical-system-path` *what*) ⇒ string
  what : symbol in `'(init-file setup-file)`

  – `'init-file` returns the path to the user-specific intialization file (containing Scheme code).
  – `'setup-file` returns the path to the file containing low-level preference settings (such as the font family mapping).

`get-panel-background`

Returns the background color of a panel (usually some shade of gray) for the current platform.

- (`get-panel-background`) $\Rightarrow$ `color%` object

`get-resource`

- (`get-resource` *section* *entry* *value* *file*) $\Rightarrow$ boolean
    section : string
    entry : string
    value : boxed string or boxed exact integer
    file = `#f` : string or `#f`

Gets a resource value from the resource database. If *file* is `#f`, the platform-specific resource database is read, as determined by `find-graphical-system-path` with `'setup-file`. (Under X, when *file* is `#f`, the user's **.Xdefaults** file is also read, or the file specified by the **XENVIRONMENT** environment variable.)

The resource value is keyed on the combination of *section* and *entry*. The `"mred"` section is reserved for PLT.

The return value is `#t` is a value is found, `#f` if it is not. The type of the value initially in the *value* box determines the way that the resource is interpreted, and *value* is filled with a new value of the same type if one is found.

See also `write-resource`.

`get-window-text-extent`

Returns the pixel size of a string drawn as a window's label or value.

See also `get-text-extent` .

- (`get-window-text-extent` *string* *font*) $\Rightarrow$ two exact non-negative integers: width and height
    string : string
    font : `font%` object

Returns the extent of the given string drawn with the given font. For information about the font used to draw a window's label or value, see `set-label-font` and `set-control-font`.

`graphical-read-eval-print-loop`

Similar to MzScheme's `read-eval-print-loop`, except that none of `read-eval-print-loop`'s configuration parameters are used (such as `current-read`) and the interaction occurs in a GUI window instead of using the current input and output ports.

Expressions entered into the graphical read-eval-print loop can be evaluated in an eventspace (and thread) that is distinct from the one implementing the `graphical-read-eval-print-loop` window (i.e., the current eventspace when `graphical-read-eval-print-loop` is called).

If no eventspace is provided, or if `#f` is provided, an evaluation eventspace is created using (`make-eventspace`) with a new custodian; the eventspace and its threads are be shut down when the user closes the `graphical-read-eval-print-loop` window. The following parameters are initialized in the created eventspace's handler thread:

- `current-output-port` — writes to the frame

- `current-error-port` — writes to the frame

- `current-input-port` — always returns `eof`

- `current-will-executor` — a new will executor

See (see parameters, §9.4 in *PLT MzScheme: Language Manual*) for more information about these parameters.

If an evalaution eventspace is provided to `graphical-read-eval-print-loop`, the parameters are not set (but evaluation output still goes to the new frame).

The keymap for the read-eval-print loop's editor is initialized by calling the current keymap initializer procedure, which is determined by the `current-text-keymap-initializer` parameter.

- (`graphical-read-eval-print-loop` *eval-eventspace*) ⇒ void
  eval-eventspace = `#f` : eventspace

`is-busy?`

- (`is-busy?`) ⇒ boolean

  Returns `#t` if a busy cursor has been installed with `begin-busy-cursor` and not removed with `end-busy-cursor`.

`label->plain-label`

- (`label->plain-label` *label*) ⇒ string
  label : string

  Strips shortcut ampersands from *label* and returns the label as it would appear on a button.

`play-sound`

- (`play-sound` *filename async?*) ⇒ boolean
  filename : string
  async? : boolean

  Plays a sound file. If *async?* is false, the function does not return until the sound completes. Otherwise, it returns immediately. The result is `#t` if the sound plays successfully, `#f` otherwise.

  Under X Windows, the play command must be defined through the user's X resources file with the resource name "mred.playcmd"; this command string is `format`ted with the input filename (so the command string should contain a "∼s" where the filename should be substituted) and executed as a shell command. the default command is `"cat ∼s > /dev/audio"`.

  Under Windows, only ".wav" files are supported.

  Under MacOS, only standard Macintosh sound files (SND) are supported.

`send-message-to-window`

- (`send-message-to-window` *x y message*) ⇒ value
  x : exact integer in [-10000, 10000]

y : exact integer in [-10000, 10000]
message : value

Finds the frontmost top-level window at (x, y) in global coordinates. If a window is there, this function calls the window's `on-message` method, providing *message* as the method's argument; the result of the function call is the result returned by the method. If no MrEd window is at the given coordinates, or if it is covered by a non-MrEd window at (x, y), `#f` is returned.

`the-clipboard`

See `clipboard<%>`.

- `the-clipboard` ⇒ `clipboard<%>` object
    Initial value : the clipboard

`write-resource`

- (`write-resource` *section* *entry* *value* *file*) ⇒ boolean
    section : string
    entry : string
    value : string or exact integer
    file = `#f` : string or `#f`

Writes a resource value to the specified resource database. If *file* is `#f`, the platform-specific resource database is read, as determined by `find-graphical-system-path` with `'setup-file`.

The resource value is keyed on the combination of *section* and *entry*. The `"mred"` section is reserved for PLT.

The return value is `#t` if the write suceeds, `#f` otherwise. (A failure indicates that the resource file cannot be written.)

If *value* is an integer outside a platform-specific range, an `exn:application:mismatch` exception is raised.

See also `get-resource`.

Part II

Drawing Toolbox

5. Drawing Toolbox Overview

Drawing in MrEd requires a **device context** (**DC**), which is an instance of the `dc<%>` interface. For example, the `get-dc` method of a canvas returns a `dc<%>` instance for drawing into the canvas window. Other kinds of DCs draw to different kinds of devices:

- `bitmap-dc%` — a **bitmap DC** draws to an offscreen bitmap.
- `post-script-dc%` — a **PostScript DC** records drawing commands to a PostScript file.
- `printer-dc%` — a **printer DC** draws to a platform-specific printer device (Windows, MacOS).

Tools that are used for drawing include the following: `pen%` objects for drawing lines and shape outlines, `brush%` objects for filling shapes, and `bitmap%` objects for storing bitmaps.

The following example creates a frame with a drawing canvas, and then draws a round, blue face with square, yellow eyes and a smiling, red mouth:

```
; Make a 300 × 300 frame
(define frame (make-object frame% "Drawing Example" #f 300 300))
; Make the drawing area
(define canvas (make-object canvas% frame))
; Get the canvas's drawing context
(define dc (send canvas get-dc))

; Make some pens and brushes
(define no-pen (make-object pen% "BLACK" 1 'transparent))
(define no-brush (make-object brush% "BLACK" 'transparent))
(define blue-brush (make-object brush% "BLUE" 'solid))
(define yellow-brush (make-object brush% "YELLOW" 'solid))
(define red-pen (make-object pen% "RED" 2 'solid))

; Define a procedure to draw a face
(define (draw-face dc)
  (send dc set-pen no-pen)
  (send dc set-brush blue-brush)
  (send dc draw-ellipse 50 50 200 200)

  (send dc set-brush yellow-brush)
  (send dc draw-rectangle 100 100 10 10)
  (send dc draw-rectangle 200 100 10 10)

  (send dc set-brush no-brush)
  (send dc set-pen red-pen)
  (let ([pi (atan 0 -1)])
    (send dc draw-arc 75 75 150 150 (* 5/4 pi) (* 7/4 pi))))
```

```
; Show the frame
(send frame show #t)
; Wait a second to let the window get ready
(sleep/yield 1)
; Draw the face (if the window is ready)
(draw-face dc)
```

The `sleep/yield` call is necessary under X because drawing to the canvas has no effect when the canvas is not shown. Although the `(send frame show #t)` expression queues a show request for the frame, the actual display of the frame and its canvas requires handling several events. The `sleep/yield` procedure pauses for a specified number of seconds, handling events while it pauses.

One second is plenty of time for the frame to show itself, but a better solution is to create a canvas with an `on-paint` method. Using `on-paint` is better for all platforms; when the canvas in the above example is resized or temporarily covered by another window, the face disappears. To ensure that the face is redrawn whenever the canvas itself is repainted, we override the canvas's `on-paint` method:

```
; Make a 300 × 300 frame
(define frame (make-object frame% "Drawing Example" #f 300 300))

; Derive a class for a canvas that calls draw-face to repaint itself
(define face-canvas%
  (class canvas% (frame)
    (inherit get-dc)
    (override [on-paint (lambda () (draw-face (get-dc)))])
    (sequence (super-init frame))))
; Make the drawing area
(define canvas (make-object face-canvas% frame))

; ... pens, brushes, and draw-face are the same as above ...

; Show the frame
(send frame show #t)
```

Suppose that `draw-face` creates a particularly complex face that takes a long time to draw. We might want to draw the face once into an offscreen bitmap, and then override `on-paint` to copy the cached bitmap image onto the canvas whenever the canvas is updated. To draw into a bitmap, we first create a `bitmap%` object, and then we create a `bitmap-dc%` to direct drawing commands into the bitmap:

```
; ... pens, brushes, and draw-face are the same as above ...

; Create a 300 × 300 bitmap
(define face-bitmap (make-object bitmap% 300 300))
; Create a drawing context for the bitmap
(define bm-dc (make-object bitmap-dc%))
; Direct the drawing context to the bitmap
(send bm-dc set-bitmap face-bitmap)
; A new bitmap's initial content is undefined, so clear it before drawing
(send bm-dc clear)

; Draw the face into the bitmap
(draw-face bm-dc)
```

```
;  Make a 300 × 300 frame
(define frame (make-object frame% "Drawing Example" #f 300 300))

;  Derive a class for a canvas that copies the bitmap to repaint itself
(define bitmap-face-canvas%
  (class canvas% (frame)
    (inherit get-dc)
    (override [on-paint (lambda () (send (get-dc) draw-bitmap face-bitmap 0 0))])
    (sequence (super-init frame))))
;  Make the drawing area
(define canvas (make-object bitmap-face-canvas% frame))

;  Show the frame
(send frame show #t)
```

For all types of DCs, the drawing origin is the top-left corner of the DC. When drawing to a window or bitmap, DC units initially correspond to pixels, but the `set-scale` method changes the scale. When drawing to a PostScript or printer device, DC units initially correspond to points (1/72 of an inch).

Drawing effects are not completely portable across platforms or across types of DC. The drawing toolbox provides tools to draw images precisely and portably, but also provides convenience tools for occasions when precision or portability is not necessary. For example, drawing with a pen of width 0 or 1 produces reliable results for all platforms and unscaled DCs, but a pen width of 2 or drawing to a scaled DC looks slightly different depending on the platform and destination.

6.   Drawing Class Reference

6.1   Class Listing

Device Contexts

```
dc<%>
  |- bitmap-dc%
  |- post-script-dc%
  |- printer-dc%
```

Drawing Tools

```
pen%
brush%
font%
color%
bitmap%
point%
region%
```

Miscellaneous

```
pen-list%
brush-list%
font-list%
font-name-directory<%>
color-database<%>
ps-setup%
```

6.2   `bitmap%`

A `bitmap%` object is a pixel-based image, either monochrome or color.

Sometimes, a bitmap object creation fails in a low-level manner. In that case, the `ok?` method returns `#f`, and the bitmap cannot be supplied to methods that consume or operate on bitmaps (otherwise, an `exn:application:mismatch` exception is raised).

- (make-object bitmap% *bits width height*) ⇒ bitmap% object
    bits : string
    width : exact integer in [1, 10000]
    height : exact integer in [1, 10000]

  Creates a monochrome bitmap from an array of bit values, where each character in *bits* specifies eight

bits, and padding bits are added so that each bitmap line starts on a character boundary. A `1` bit value indicates black, and `0` indicates white. If *width* times *height* is larger than 8 times the length of *bits*, an `exn:application:mismatch` exception is raised.

- (make-object bitmap% *width height monochrome?*) ⇒ bitmap% object
    width : exact integer in [1, 10000]
    height : exact integer in [1, 10000]
    monochrome? = #f : boolean

Creates a new bitmap. If *monochrome?* is `#f`, the bitmap matches the display depth of the screen. The initial content of the bitmap is undefined.

- (make-object bitmap% *name kind*) ⇒ bitmap% object
    name : string
    kind = 'unknown : symbol in '(unknown gif jpeg xbm xpm bmp pict)

Creates a bitmap from a file, where *kind* specifies the kind of image file. See `load-file` for details.

get-depth

Gets the color depth of the bitmap. See also `is-color?`.

- (send *a-bitmap* get-depth) ⇒ exact non-negative integer

get-height

Gets the height of the bitmap in pixels.

- (send *a-bitmap* get-height) ⇒ exact integer in [1, 10000]

get-width

Gets the width of the bitmap in pixels.

- (send *a-bitmap* get-width) ⇒ exact integer in [1, 10000]

is-color?

Returns `#f` if the bitmap is monochrome, `#t` otherwise.

- (send *a-bitmap* is-color?) ⇒ boolean

load-file

Loads a bitmap from a file. If the bitmap is in use by a `bitmap-dc%` object or a control, the bitmap file is not loaded.

- (send *a-bitmap* load-file *name kind*) ⇒ boolean
    name : string
    kind = 'unknown : symbol in '(unknown gif jpeg xbm xpm bmp pict)

The *kind* parameter specifies the file's format:

- – `'unknown` — examine the file to determine its format
- – `'gif` — load a GIF bitmap file (X, Windows, MacOS)
- – `'jpeg` — load a JPEG bitmap file (X, Windows, MacOS)
- – `'xbm` — load an X bitmap file (X, Windows, MacOS); creates a monochrome bitmap
- – `'xpm` — load an XPM bitmap file (X, Windows, MacOS)
- – `'bmp` — load a Windows bitmap file (X, Windows, MacOS)
- – `'pict` — load a PICT bitmap file (MacOS)

An XBM image is always loaded as a monochrome bitmap. An image in any other format is always loaded as a bitmap that matches the depth of the screen.

`ok?`

Returns `#t` if the bitmap is usable (created or changed successfully). If `#f` is returned, the bitmap cannot be supplied to methods that consume or operate on bitmaps (otherwise, an `exn:application:mismatch` exception is raised).

- (send *a-bitmap* `ok?`) ⇒ boolean

`save-file`

Saves a bitmap in the named file.

- (send *a-bitmap* `save-file` *name kind*) ⇒ boolean
    name : string
    kind : symbol in `'(xbm xpm bmp pict)`

  The *kind* argument determined the type of file that is created, one of:
    - – `'xbm` — save an X bitmap file (X, Windows, MacOS)
    - – `'xpm` — save an XPM bitmap file (X, Windows, MacOS)
    - – `'bmp` — save a Windows bitmap file (Windows)
    - – `'pict` — save a PICT bitmap file (MacOS)

6.3 `bitmap-dc%`

Implements: `dc<%>`

A `bitmap-dc%` object allows drawing direclty into a bitmap. A `bitmap%` object must be supplied at initialization or installed into a bitmap DC using `set-bitmap` before any other method of the DC is called. If a `bitmap-dc%` method is called before a bitmap is selected, the method call is ignored.

- (make-object `bitmap-dc%` *bm*) ⇒ `bitmap-dc%` object
    bm : `bitmap%` object or `#f`

  Creates a new memory DC. If *bm* is not `#f`, it is installed into the DC so that drawing commands on the DC draw to *bm*. Otherwise, no bitmap is installed into the DC and `set-bitmap` must be called before any other method of the DC is called.

`get-bitmap`

Gets the bitmap currently installed in the DC, or `#f` if no bitmap is installed. See `set-bitmap` for more information.

- (send *a-bitmap-dc* get-bitmap) ⇒ `bitmap%` object or `#f`

`get-pixel`

Gets the current color of a pixel in the bitmap.

Under X, interleaving drawing commands with `get-pixel` calls (for the same `bitmap-dc%` object) incurs a substantial performance penalty.

- (send *a-bitmap-dc* get-pixel *x y color*) ⇒ boolean
    x : real number
    y : real number
    color : `color%` object

  Fills *color* with the color of the current pixel at position (x, y) in the drawing context. If the color is successfully obtained, the return value is `#t`, otherwise the result is `#f`.

`set-bitmap`

Installs a bitmap into the DC, so that drawing operations on the bitmap DC draw to the bitmap. A bitmap is removed from a DC by setting the bitmap to `#f`.

A bitmap can be selected into at most one bitmap DC, and only when it is not used by a control (as a label) or in a `pen%` or `brush%` (as a stipple). If the argument to `set-bitmap` is already in use by another DC, a control, a `pen%`, or a `brush%`, an `exn:application:mismatch` exception is raised.

- (send *a-bitmap-dc* set-bitmap *bitmap*) ⇒ void
    bitmap : `bitmap%` object or `#f`

`set-pixel`

Sets a pixel in the bitmap.

Under X, interleaving drawing commands with `set-pixel` calls (for the same `bitmap-dc%` object) incurs a substantial performance penalty.

- (send *a-bitmap-dc* set-pixel *x y color*) ⇒ void
    x : real number
    y : real number
    color : `color%` object

6.4   `brush%`

A brush is a drawing tool with a color and a style that is used for filling in areas, such as the interior of a rectangle or ellipse. On a monochrome display, all non-white brushes are drawn as black.

In addition to its color and style, a brush can have a stipple bitmap. Painting with a stipple brush is similar to calling `draw-bitmap` with the stipple bitmap in the filled region.

A brush's style is one of the following:

- `'transparent` — Draws with no effect (on the interior of the drawn shape).

- **'solid** — Draws using the brush's color. If a monochrome stipple is installed into the brush, black pixels from the stipple are transferred to the destination using the brush's color, and white pixels from the stipple are not transferred.

- **'opaque** — Same as **'solid**, except when a monochrome stipple is installed; in that case, white pixels from the stipple are transferred to the destination using the destination's background color.

- **'xor** — If a color stipple is installed, **'xor** is treated as **'solid**. Otherwise, the brush's color or colored (monochrome) stipple is xored with existing destination pixel values. The **'xor** mapping is unspecified for arbitrary color combinations, but the mapping provides two guarantees:
  - Black-and-white drawing to a color or monochrome destination always works as expected: black xor white = black, white xor black = black, black xor black = white, and white xor white = white.
  - Performing the same drawing operation twice in a row with **'xor** is equivalent to a no-op.

- The following modes correspond to built-in stipples drawn in **'solid** mode:
  - **'bdiagonal-hatch** — diagonal lines, top-left to bottom-right
  - **'crossdiag-hatch** — crossed diagonal lines
  - **'fdiagonal-hatch** — diagonal lines, top-right to bottom-left
  - **'cross-hatch** — crossed horizontal and vertical lines
  - **'horizontal-hatch** — horizontal lines
  - **'vertical-hatch** — vertical lines

  However, when a specific stipple is installed into the brush, the above modes are ignored and **'solid** is used, instead.

To draw outline shapes (such as unfilled boxes and ellipses), use the **'transparent** brush style. See `set-style` for more information about styles.

To avoid creating multiple brushes with the same characteristics, use the global `brush-list%` object `the-brush-list`.

- `(make-object brush%)` ⇒ `brush%` object

  Creates a solid black brush.

- `(make-object brush%` *color style*`)` ⇒ `brush%` object
  color : `color%` object
  style : symbol   in   '(transparent solid opaque xor bdiagonal-hatch
          crossdiag-hatch fdiagonal-hatch cross-hatch horizontal-hatch
          vertical-hatch)
  Creates a brush with the given color and style.

- `(make-object brush%` *color-name style*`)` ⇒ `brush%` object
  color-name : string
  style : symbol   in   '(transparent solid opaque xor bdiagonal-hatch
          crossdiag-hatch fdiagonal-hatch cross-hatch horizontal-hatch
          vertical-hatch)
  Creates a brush with the given color and style, where the color is specified using a name; see `color-database<%>` for information about color names. If the name is not known, the brush's color is set to black.

get-color

Returns the brush's color.

- (send *a-brush* `get-color`) ⇒ `color%` object

get-stipple

Gets the stipple bitmap, or `#f` if the brush has no stipple.

- (send *a-brush* `get-stipple`) ⇒ `bitmap%` object or `#f`

get-style

Returns the brush's style. See `brush%` for information about brush styles.

- (send *a-brush* `get-style`) ⇒ symbol in `'(transparent solid opaque xor bdiagonal-hatch crossdiag-hatch fdiagonal-hatch cross-hatch horizontal-hatch vertical-hatch)`

set-color

Sets the brush's color.

A brush cannot be modified if it was obtained from a `brush-list%` or while it is selected into a drawing context.

- (send *a-brush* `set-color` *color*) ⇒ void
    color : `color%` object

    Sets the brush's color to match the given color.

- (send *a-brush* `set-color` *color-name*) ⇒ void
    color-name : string

    Set's the brushes color to *color-name* if the name is known; see `color-database<%>` for information about color names.

- (send *a-brush* `set-color` *red green blue*) ⇒ void
    red : exact integer in [0, 255]
    green : exact integer in [0, 255]
    blue : exact integer in [0, 255]

    Sets the RGB values of the brush's color.

set-stipple

Sets or removes the stipple bitmap, where `#f` removes the stipple. See `brush%` for information about drawing with stipples.

A bitmap cannot be used as a stipple if it is selected into a `bitmap-dc%` object; if the given bitmap is selected into a `bitmap-dc%` object, an `exn:application:mismatch` exception is raised. A brush cannot be modified if it was obtained from a `brush-list%` or while it is selected into a drawing context.

- (send *a-brush* `set-stipple` *bitmap*) ⇒ void
    bitmap : `bitmap%` object or `#f`

`set-style`

Sets the brush's style. See `brush%` for information about the possible styles.

A brush cannot be modified if it was obtained from a `brush-list%` or while it is selected into a drawing context.

- (send *a-brush* `set-style` *style*) $\Rightarrow$ void
    style : symbol   in   '(transparent solid opaque xor bdiagonal-hatch
          crossdiag-hatch fdiagonal-hatch cross-hatch horizontal-hatch
          vertical-hatch)

6.5   `brush-list%`

A `brush-list%` object maintains a list of `brush%` objects to avoid creating brushes repeatedly. A `brush%` object in a brush list cannot be mutated.

A global brush list, `the-brush-list`, is created automatically.

- (`make-object` `brush-list%`) $\Rightarrow$ `brush-list%` object
    Creates an empty brush list.

`find-or-create-brush`

Finds a brush of the given specification, or creates one and adds it to the list.

- (send *a-brush-list* `find-or-create-brush` *color style*) $\Rightarrow$ `brush%` object
    color : `color%` object
    style : symbol   in   '(transparent solid opaque xor bdiagonal-hatch
          crossdiag-hatch fdiagonal-hatch cross-hatch horizontal-hatch
          vertical-hatch)
    See `brush%`.
- (send *a-brush-list* `find-or-create-brush` *color-name style*) $\Rightarrow$ `brush%` object or `#f`
    color-name : string
    style : symbol   in   '(transparent solid opaque xor bdiagonal-hatch
          crossdiag-hatch fdiagonal-hatch cross-hatch horizontal-hatch
          vertical-hatch)
    See `brush%`.
    The return value is `#f` when no color matching *color-name* can be found in the color database.

6.6   `color%`

A color is an object representing a red-green-blue (RGB) combination of primary colors, and is used to determine drawing colors. Each red, green, or blure component of the color is in the range 0 to 255, inclusive. For example, (0, 0, 0) is black, (255, 255, 255) is white, and (255, 0, 0) is red.

See `color-database<%>` for infomation about obtaining a color object using a color name.

- (`make-object` `color%` *red green blue*) $\Rightarrow$ `color%` object
    red : exact integer in [0, 255]

        green : exact integer in [0, 255]
        blue : exact integer in [0, 255]

    Creates a new color with the given RGB values.

- `(make-object color%` *color-name*`)` ⇒ `color%` object
    color-name : `string`

    Creates a new color by name, using `the-color-database`. (See `color-database<%>` for more information.)

blue

Returns the blue component of the color.

- `(send` *a-color* `blue)` ⇒ exact integer in [0, 255]

copy-from

Copies the RGB values of another color object to this one, returning this object as the result.

- `(send` *a-color* `copy-from` *src*`)` ⇒ `color%` object
    src : `color%` object

green

Returns the green component of the color.

- `(send` *a-color* `green)` ⇒ exact integer in [0, 255]

ok?

Returns `#t` if the color object is valid.

- `(send` *a-color* `ok?)` ⇒ boolean

red

Returns the red component of the color.

- `(send` *a-color* `red)` ⇒ exact integer in [0, 255]

set

Sets the three (red, green, and blue) component values of the color.

- `(send` *a-color* `set` *red green blue*`)` ⇒ void
    red : exact integer in [0, 255]
    green : exact integer in [0, 255]
    blue : exact integer in [0, 255]

6.7   `color-database<%>`

The global `the-color-database` object is an instance of `color-database<%>`. It maintains a database of standard RGB colors for a predefined set of named colors (such as "black" and "light grey").

The following colors are in the database:

> aquamarine, black, blue, blue violet, brown, cadet blue, coral, cornflower blue, cyan, dark gray, dark green, dark olive green, dark orchid, dark slate blue, dark slate gray, dark turquoise, dim gray, firebrick, forest green, gold, goldenrod, gray, green, green yellow, indian red, khaki, light blue, light gray, light steel blue, lime green, magenta, maroon, medium aquamarine, medium blue, medium forest green, medium goldenrod, medium gray, medium orchid, medium sea green, medium slate blue, medium spring green, medium turquoise, medium violet red, midnight blue, navy, orange, orange red, orchid, pale green, pink, plum, purple, red, salmon, sea green, sienna, sky blue, slate blue, spring green, steel blue, tan, thistle, turquoise, violet, violet red, wheat, white, yellow, yellow green

The names are not case-sensitive.

See also `color%`.

find-color

Finds a color by name (character case is ignored). If no color is found for the name, `#f` is returned.

- (send *a-color-database* `find-color` *color-name*) $\Rightarrow$ `color%` object or `#f`
    color-name : `string`

  See `color-database<%>` for the list of color names.

6.8   `dc<%>`

A `dc<%>` object is a device context for drawing graphics and text. It represents output devices in a generic way; e.g., a canvas has a device context, as does a printer.

The drawing methods, such as `draw-rectangle`, accept real number values as arguments, but the results are only well-defined when the drawing coordinates are in the range `-16383` to `16383`. This restriction applies to the coordinates both before and after offsets and scaling factors are applied.

clear

Clears the drawing region (fills it with the current background brush).

- (send *a-dc* `clear`) $\Rightarrow$ `void`

draw-arc

Draws an arc. The current pen is used for the arc and the current brush for filling a wedge.

If both the pen and brush are non-transparent, the wedge is filled with the brush before the arc is drawn with the pen. The wedge and arc meet so that no space is left between them, but the precise overlap between the

wedge and arc is platform- and size-specific. Thus, the regions drawn by the brush and pen may partially overlap.

- (send *a-dc* draw-arc *x y width height start-radians end-radians*) ⇒ void
  x : real number
  y : real number
  width : non-negative real number
  height : non-negative real number
  start-radians : real number
  end-radians : real number

Draws a counter-clockwise circular arc, a part of the ellipse inscribed in the rectangle specified by *x* (left), *y* (top), *width*, and *height*. The arc starts at the angle specified by *start-radians* (0 is 3 o'clock) and continues counter-clockwise to *end-radians*. If *start-radions* and *end-radians* are the same, a full ellipse is drawn.

If the current brush is not transparent, it is used to fill the wedge bounded by the arc plus lines (not drawn) extending to the center of the inscribed ellipse.

Restrictions on the magnitude of drawing coordinates are described with dc<%>.

draw-bitmap

Displays a bitmap. For color bitmaps, the drawing style and color arguments are ignored. For monochrome bitmaps, draw-bitmap uses the style and color arguments in the same way that a brush uses its style and color settings to draw a monochrome stipple (see brush% for more information).

See also draw-bitmap-section.

The current brush, current pen, and current text settings for the DC have no effect on how the bitmap is drawn.

- (send *a-dc* draw-bitmap *source xdest ydest style color*) ⇒ boolean
  source : bitmap% object
  xdest : real number
  ydest : real number
  style = 'solid : symbol in '(solid opaque xor)
  color = black : color% object

The *xdest* and *ydest* arguments are in DC coodinates and may be scaled, but the source bitmap is never scaled as it is copied. Thus, if the DC has a scaling factor of 2, the destination width in DC units is *width*/2.

Restrictions on the magnitude of drawing coordinates are described with dc<%>.

draw-bitmap-section

Display part of a bitmap. See also draw-bitmap.

- (send *a-dc* draw-bitmap-section *source xdest ydest xsrc ysrc width height style color*) ⇒ boolean
  source : bitmap% object
  xdest : real number
  ydest : real number
  xsrc : real number
  ysrc : real number

>   *width* : non-negative real number
>   *height* : non-negative real number
>   *style* = 'solid : symbol in '(solid opaque xor)
>   *color* = black : color% object

The *xsrc*, *xsrc*, *width*, and *height* arguments specify a rectangle in the source bitmap to copy to this device context.

See draw-bitmapfor information about *xdest*, *ydest*, *style* and *color*.

draw-ellipse

Draws an ellipse contained in a rectangle. The current pen is used for the outline and the current brush for filling the shape.

If both the pen and brush are non-transparent, the ellipse is filled with the brush before the outline is drawn with the pen. The filling and outline meet so that no space is left between them, but the precise overlap between the filling and outline is platform- and size-specific. Thus, the regions drawn by the brush and pen may partially overlap.

-   (send *a-dc* draw-ellipse *x y width height*) ⇒ void
>   *x* : real number
>   *y* : real number
>   *width* : non-negative real number
>   *height* : non-negative real number

    Draws an ellipse that fits within a rectangle with the given top-left corner and size.

    Restrictions on the magnitude of drawing coordinates are described with dc<%>.

draw-line

Draws a line from one point to another. The line covers both the start and end points. The current pen is used for drawing the line.

-   (send *a-dc* draw-line *x1 y1 x2 y2*) ⇒ void
>   *x1* : real number
>   *y1* : real number
>   *x2* : real number
>   *y2* : real number

    Restrictions on the magnitude of drawing coordinates are described with dc<%>.

draw-lines

Draws multiple connected lines. The current pen is used for drawing the lines.

-   (send *a-dc* draw-lines *points xoffset yoffset*) ⇒ void
>   *points* : list of point% objects
>   *xoffset* = 0 : real number
>   *yoffset* = 0 : real number

    Draws lines using a list of *points*, adding *xoffset* and *yoffset* to each point.

    Restrictions on the magnitude of drawing coordinates are described with dc<%>.

`draw-point`

Plots a single point using the current pen.

- (send *a-dc* `draw-point` *x y*) ⇒ void
    x : real number
    y : real number

  Restrictions on the magnitude of drawing coordinates are described with `dc<%>`.

`draw-polygon`

Draws and paints a polygon from a list of points. The current pen is used for drawing the outline, and the current brush for filling the shape.

If both the pen and brush are non-transparent, the polygon is filled with the brush before the outline is drawn with the pen. The filling and outline meet so that no space is left between them, but the precise overlap between the filling and outline is platform- and shape-specific. Thus, the regions drawn by the brush and pen may partially overlap.

- (send *a-dc* `draw-polygon` *points xoffset yoffset fill-style*) ⇒ void
    points : list of `point%` objects
    xoffset = 0 : real number
    yoffset = 0 : real number
    fill-style = '`odd-even` : symbol in '(`odd-even winding`)

  Draw a filled polygon using a list of *points*, adding *xoffset* and *yoffset* to each point. The ploygon is automatically closed, so the first and last point can be different.

  The *fill-style* argument specifies the fill rule: '`odd-even` or '`winding`.

  Restrictions on the magnitude of drawing coordinates are described with `dc<%>`.

`draw-rectangle`

Draws a rectangle. The current pen is used for the outline and the current brush for filling the shape.

If both the pen and brush are non-transparent, the rectangle is filled with the brush before the outline is drawn with the pen. When the pen is size 0 or 1, the filling precisely overlaps the entire outline. As a result, if a rectangle is drawn with a size-0 or size-1 '`xor pen%` and an '`xor brush%`, the outline is xored twice (first by the brush, then by the pen), leaving it unchanged.

- (send *a-dc* `draw-rectangle` *x y width height*) ⇒ void
    x : real number
    y : real number
    width : non-negative real number
    height : non-negative real number

  Draws a rectangle with the given top-left corner and size.

  Restrictions on the magnitude of drawing coordinates are described with `dc<%>`.

`draw-rounded-rectangle`

Draws a rectangle with rounded corners. The current pen is used for the outline and the current brush for filling the shape.

If both the pen and brush are non-transparent, the rectangle is filled with the brush before the outline is drawn with the pen. The filling and outline meet so that no space is left between them, but the precise overlap between the filling and outline is platform- and size-specific. Thus, the regions drawn by the brush and pen may partially overlap.

- (send *a-dc* `draw-rounded-rectangle` *x y width height radius*) ⇒ void
    x : real number
    y : real number
    width : non-negative real number
    height : non-negative real number
    radius = 20 : real number

Draws a rectangle with the given top-left corner, and with the given size. The corners are quarter-circles using the given radius.

If *radius* is positive, the value is used as the radius of the rounded corner. If *radius* is negative, the absolute value is used as the the *proportion* of the smallest dimension of the rectangle. If *radius* is more than half of *width* or *height*, the resulting image is undefined.

Restrictions on the magnitude of drawing coordinates are described with dc<%>.

`draw-spline`

Draws a three-point spline using the current pen.

- (send *a-dc* `draw-spline` *x1 y1 x2 y2 x3 y3*) ⇒ void
    x1 : real number
    y1 : real number
    x2 : real number
    y2 : real number
    x3 : real number
    y3 : real number

Draws a spline from $(x1, y1)$ to $(x3, y3)$ using $(x2, y2)$ as the control point.

Restrictions on the magnitude of drawing coordinates are described with dc<%>.

`draw-text`

Draws a text string at a specified point, using the current text font, and the current text foreground and background colors. The specified point is used as the starting top-left point for drawing characters (e.g, if "W" is drawn, the point is roughly the location of the top-left pixel in the "W").

See `get-text-extent` for information on the size of the drawn text.

See also `set-text-foreground`, `set-text-background`, and `set-text-mode`.

The current brush and current pen settings for the DC have no effect on how the text is drawn.

- (send *a-dc* `draw-text` *text x y big-chars?   offset*) ⇒ void
    text : string
    x : real number
    y : real number
    big-chars? = #f : boolean
    offset = 0 : exact non-negative integer

Restrictions on the magnitude of drawing coordinates are described with `dc<%>`.

The *text* string is drawn starting from the *offset* character, and continuing until the end of *text* or the first null character

If *big-chars?* is `#t`, then *text* is interpreted as a Unicode string instead of an ASCII string; however, Unicode strings are not yet fully supported.

end-doc

Ends a document, relevant only when drawing to a printer or PostScript device.

    - (send *a-dc* `end-doc`) ⇒ void

end-page

Ends a single page, relevant only when drawing to a printer or PostScript device.

    - (send *a-dc* `end-page`) ⇒ void

get-background

Gets the color used for painting the background. See also `set-background`.

    - (send *a-dc* `get-background`) ⇒ `color%` object

get-brush

Gets the current brush. See also `set-brush`.

    - (send *a-dc* `get-brush`) ⇒ `brush%` object

get-char-height

Gets the height of a character using the current font.

    - (send *a-dc* `get-char-height`) ⇒ non-negative real number

get-char-width

    - (send *a-dc* `get-char-width`) ⇒ non-negative real number

    Gets the average width of a character using the current font.

get-clipping-region

Gets the current clipping region, returning `#f` if the drawing context is not clipped (i.e., the clipping region is the entire drawing region).

    - (send *a-dc* `get-clipping-region`) ⇒ `region%` object or `#f`

get-font

Gets the current font. See also set-font.

- (send *a-dc* get-font) ⇒ font% object

get-pen

Gets the current pen. See also set-pen.

- (send *a-dc* get-pen) ⇒ pen% object

get-size

Gets the size of the destination drawing area. For a dc<%> object obtained from a canvas<%>, this is the (virtual client) size of the desintation window; for a bitmap-dc% object, this is the size of the selected bitmap (or 0 if no bitmap is selected); for a post-script-dc% or printer-dc% device context, this gets the horizontal and vertical size of the drawing area.

- (send *a-dc* get-size) ⇒ two non-negative real numbers: width and height

get-text-background

Gets the current text background color. See also set-text-background.

- (send *a-dc* get-text-background) ⇒ color% object

get-text-extent

Gets the dimensions of a string drawn into this device context. The result is four real numbers:

- the total width of the text (depends on both the font and the text);

- the total height of the font (depends only on the font);

- the distance from the baseline of the font to the bottom of the descender (included in the height, depends only on the font); and

- extra vertical space added to the font by the font designer (included in the height, and often zero; depends only on the font).

The returned width and height define a rectangle is that guaranteed to contain the text string when it is drawn, but the fit is not necessarily tight. Some undefined number of pixels on the left, right, top, and bottom of the drawn string may be "whitespace," depending on the whims of the font designer and the platform-specific font-scaling mechanism.

- (send *a-dc* get-text-extent *string font big-chars? offset*) ⇒ four non-negative real numbers: width, height, descent, and space
    string : string

    font = #f : font% object or #f
    big-chars? = #f : boolean
    offset = 0 : exact non-negative integer

Returns the size of *string* at it would be drawn in the device context, starting from the *offset* character of *string*, and continuing until the end of *string* or the first null character. The *font* argument specifies the font to use in measuring the text; if it is `#f`, the current font of the drawing area is used. (See also `set-font`.)

If *big-chars?* is `#t`, then *string* is interpreted as a string or Unicode or 16-bit characters instead of an ASCII string, but such strings are not yet fully supported.

get-text-foreground

Gets the current text foreground color. See also `set-text-foreground`.

- (send *a-dc* get-text-foreground) ⇒ color% object

get-text-mode

Reports how text is drawn; see `set-text-mode` .

- (send *a-dc* get-text-mode) ⇒ symbol in '(solid transparent)

ok?

Returns `#t` if the drawing context is useable.

- (send *a-dc* ok?) ⇒ boolean

set-background

Sets the background color for drawing in this object. On a monochrome display, all non-black colors are treated as white.

- (send *a-dc* set-background *color*) ⇒ void
    color : color% object

set-brush

Sets the current brush for drawing in this object. While a brush is selected into a drawing context, it cannot be modified.

- (send *a-dc* set-brush *brush*) ⇒ void
    brush : brush% object

set-clipping-rect

Sets the clipping region to a rectangular region.

See also `set-clipping-region` and `get-clipping-region`.

- (send *a-dc* `set-clipping-rect` *x y width height*) ⇒ void
    x : real number
    y : real number
    width : non-negative real number
    height : non-negative real number

Restrictions on the magnitude of drawing coordinates are described with `dc<%>`.

set-clipping-region

Sets the clipping region for the drawing area, turning off all clipping within the drawing region if `#f` is provided.

The clipping region must be reset after changing a `dc<%>` object's origin or scale (unless it is `#f`); see `region%` for more information.

See also `set-clipping-rect` and `get-clipping-region`.

- (send *a-dc* `set-clipping-region` *rgn*) ⇒ void
    rgn : region% object or #f

set-font

Sets the current font for drawing text in this object.

- (send *a-dc* `set-font` *font*) ⇒ void
    font : font% object

set-origin

Sets the device origin, i.e., the location in device coordinates of (0, 0) in logical coordinates.

Changing a `dc<%>` object's origin or scale invalidates `region%` objects associated with the device context. See `region%` for more information.

Restrictions on the magnitude of drawing coordinates are described with `dc<%>`.

- (send *a-dc* `set-origin` *x y*) ⇒ void
    x : real number
    y : real number

set-pen

Sets the current pen for this object.

The current pen does not affect text drawing; see also `set-text-foreground`.

While a pen is selected into a drawing context, it cannot be modified.

- (send *a-dc* `set-pen` *pen*) ⇒ void
    pen : pen% object

set-scale

Sets a scaling factor that maps logical coordinates to device coordinates.

Changing a dc<%> object's origin or scale invalidates region% objects associated with the device context. See region% for more information.

Restrictions on the magnitude of drawing coordinates are described with dc<%>.

- (send *a-dc* set-scale *x-scale y-scale*) ⇒ void
    x-scale : non-negative real number
    y-scale : non-negative real number

set-text-background

Sets the current text background color for this object. The text background color is painted behind text that is drawn with draw-text, but only for the 'solid text mode (see set-text-mode).

On a monochrome display, all non-white colors are treated as black.

- (send *a-dc* set-text-background *color*) ⇒ void
    color : color% object

set-text-foreground

- (send *a-dc* set-text-foreground *color*) ⇒ void
    color : color% object

    Sets the current text foreground color for this object, used for drawing text with draw-text.

    On a monochrome display, all non-black colors are treated as white.

set-text-mode

Determines how text is drawn:

- 'solid — Before text is drawn, the destination area is filled with the text background color (see set-text-background).

- 'transparent — Text is drawn directly over any existing image in the destination, as if overlaying text written on transparent film.

- (send *a-dc* set-text-mode *mode*) ⇒ void
    mode : symbol in '(solid transparent)

start-doc

Starts a document, relevant only when drawing to a printer or PostScript device.

- (send *a-dc* start-doc *message*) ⇒ boolean
    message : string

    For some platforms, the *message* string is displayed in a dialog until end-doc is called.

`start-page`

Starts a page, relevant only when drawing to a printer or PostScript device.

- (send *a-dc* `start-page`) $\Rightarrow$ void

`try-color`

Detrmines the actual color used for drawing requests with the given color.

- (send *a-dc* `try-color` *try* *result*) $\Rightarrow$ void
  try : `color%` object
  result : `color%` object

  The *result* color is set to the RGB values that are actually produced for this device context to draw the color *try*.

6.9   `font%`

A font is an object which determines the appearance of text, primarily when drawing text to a device context. A font is determined by six properties:

- size — The size of the text in logical drawing units (usually points, 1/72 inch).

- family — A platform- and device-indepedent font designation. The families are:
  - `'default`
  - `'decorative`
  - `'roman`
  - `'script`
  - `'swiss`
  - `'modern`   (fixed width)
  - `'symbol`   (Greek letters)
  - `'system`   (used to draw control labels)

- face — A string face name, such as `"Times"` (under Windows and MacOS) or `"-*-times"` (under X). The format and meaning of a face name is platform- and device-specific. If a font's face name is `#f`, then the font's appearance depends only on the family. If a face is provided but no mapping is available for the face name (for a specific platform or device), then the face name is ignored and the family is used. See `font-name-directory<%>` for information about how face names are mapped for drawing text.

- style — The slant style of the font, one of:
  - `'normal`
  - `'slant` (Windows, MacOS: same as `'italic`; X: tries `'italic` if `'slant` font does not exist)
  - `'italic` (X: tries `'slant` if `'italic` font does not exist)

- weight — The weight of the font, one of:
  - `'normal`
  - `'light`
  - `'bold`

- underlining — `#t` for underlined, `#f` for plain

To avoid creating multiple fonts with the same characteristics, use the global `font-list%` object `the-font-list`.

See also `font-name-directory<%>`.

- (make-object font%) ⇒ font% object

    Creates an instance of the default font.

- (make-object font% *size family style weight underline*) ⇒ font% object
    size : exact integer in [1, 255]
    family : symbol   in   '(default decorative roman script swiss modern symbol system)
    style : symbol in '(normal italic slant)
    weight : symbol in '(normal bold light)
    underline = #f : boolean

    Creates a font with a family, but no face name.

    See `font%` for information about *family*, *style*, and *weight*. `font-name-directory<%>`.

- (make-object font% *size face family style weight underline*) ⇒ font% object
    size : exact integer in [1, 255]
    face : string
    family : symbol   in   '(default decorative roman script swiss modern symbol system)
    style : symbol in '(normal italic slant)
    weight : symbol in '(normal bold light)
    underline = #f : boolean

    See `font%` for information about *family*, *style*, and *weight*. See also `font-name-directory<%>` for information about the way *face* is interpreted for drawing text on various platforms and devices. When a platform- or device-specific interpretation of *face* is not availaable, the *family* is used to draw text.

`get-face`

Gets the font's face name, or `#f` if none is specified.

- (send *a-font* get-face) ⇒ string or #f

`get-family`

Gets the font's family. See `font%` for information about families.

- (send *a-font* get-family) ⇒ symbol in '(default decorative roman script swiss modern symbol system)

`get-font-id`

Gets the font's ID, for use with a `font-name-directory<%>`. The ID is determined by the font's face and family specifications, only.

- (send *a-font* get-font-id) ⇒ exact integer

`get-point-size`

Gets the font's size (roughly the height) in logical units.

(The method's name is misleading. When this font is used with a particular drawing context, one logical unit may or may not correspond to 1/72 inch.)

- (send *a-font* get-point-size) $\Rightarrow$ exact integer in [1, 255]

`get-style`

Gets the font's slant style. See `font%` for information about styles.

- (send *a-font* get-style) $\Rightarrow$ symbol in '(normal italic slant)

`get-underlined`

Returns `#t` if the font is underlined or `#f` otherwise.

- (send *a-font* get-underlined) $\Rightarrow$ boolean

`get-weight`

Gets the font's weight. See `font%` for information about weights.

- (send *a-font* get-weight) $\Rightarrow$ symbol in '(normal bold light)

6.10   `font-list%`

A `font-list%` object maintains a list of `font%` objects to avoid repeatedly creating fonts.

A global font list, `the-font-list`, is created automatically.

- (make-object font-list%) $\Rightarrow$ font-list% object
  Creates an empty font list.

`find-or-create-font`

Finds an existing font in the list or creates a new one (that is automatically added to the list).

- (send *a-font-list* find-or-create-font *size family style weight underline*) $\Rightarrow$ font% object
    size : exact integer in [1, 255]
    family : symbol   in   '(default decorative roman script swiss modern symbol
            system)
    style : symbol in '(normal italic slant)
    weight : symbol in '(normal bold light)
    underline = `#f` : boolean
  See `font%` for information about *family*, *style*, and *weight*.

- (send *a-font-list* `find-or-create-font` *size face family style weight underline*) ⇒ void
    size : exact integer in [1, 255]
    face : string
    family : symbol   in   '(default decorative roman script swiss modern symbol
            system)
    style : symbol in '(normal italic slant)
    weight : symbol in '(normal bold light)
    underline = `#f` : boolean

See `font%` for information about *family*, *style*, and *weight*. See also `font-name-directory<%>` about the use of *face*.

6.11 `font-name-directory<%>`

There is one `font-name-directory<%>` object: `the-font-name-directory`. It implements the mapping from font specifications (face, family, style, and weight) to information for rendering text on a specific device. The mapping is different for each platform. For example, when drawing to a bitmap in Windows, the rendering information is simply the name of a Windows font. When drawing to a PostScript file, the rendering information is a PostScript font name, which encapsulates the style and weight. When drawing to a bitmap in X, the rendering information is an X font string, which encapsulates the style and weight, parameterized over the size (using a "%d" placeholder).

Programmers rarely need to directly invoke methods of `the-font-name-directory`. It is used automatically when drawing text to a `dc<%>` object. Nevertheless, `the-font-name-directory` is available so that programmers can query or modify the mapping manually. A programmer may also need to understand how the face-and-family mapping works.

To extract mapping information from `the-font-name-directory`, first obtain a **font ID**, which is an index based on a family and and optional face string. Font IDs ar returned by `find-or-create-font-id` and `get-font-id` . A Font ID can be combined with a weight and style to obtain a specific mapping value via `get-screen-name` or `get-post-script-name`.

For a family without a face string, the corresponding font ID has a useful built-in mapping for every platform and device. For a family with a face string, `the-font-name-directory` interprets the string (in a platform-specific way) to generate a mapping for "screen" drawing (to a canvas's `dc<%>`, a `bitmap-dc%`, or a `printer-dc%`). When drawing to a `post-script-dc%` object, the face-specific mapping defaults to the family's mapping.

Under Windows and MacOS, a face name is interpreted simply as a system font name for drawing to the screen, bitmap, or printer. The mapping succeeds if the system provides a font with the given name, and fails otherwise. For example, under Windows, `"MS Sans Serif"` maps to the font that is typically used for button labels.

Under X, a face name has a more complex interpretation:

- If the string begins with `"+"`, then the remainder of the string is interpreted as an X font name. These names are usually long, such as `"+-b&h-lucidatypewriter-medium-r-normal-sans-24-240-75-75-m-140-iso8859-1"`. As usual for X font names, aterisks may appear in the string as wildcards. Furthermore, the size of the font can be parametered by using `"%d"` in the place of a specific size.

- A string of the form `"-provider-font"` is equivalent to `"+-provider-font-weight-style-normal-*-*-%d-*-*-*-*-*-*"`, where *weight* is either `medium`, `light` or `bold` (depending on the requested weight mapping) and *style* is either `r`, `i`, or `i` (depending on the requested style mapping).

- A string of the form `"-font"` is equivalent to `"-*-font"`.

- A string of any orther format is interpreted as an X font name, optionally parameterized with `"%d"`.

find-family-default-font-id

Gets the font ID reprsenting the default font for a family. See `font%` for information about font families.

- (send *a-font-name-directory* `find-family-default-font-id` *family*) ⇒ exact integer
  family : symbol   in   `'(default decorative roman script swiss modern symbol system)`

find-or-create-font-id

Gets the face name for a font ID, initializing the mapping for the face name if necessary.

Font ID are useful only as mapping indices for `the-font-name-directory`.

- (send *a-font-name-directory* `find-or-create-font-id` *name family*) ⇒ exact integer
  name : string
  family : symbol   in   `'(default decorative roman script swiss modern symbol system)`

get-face-name

Gets the face name for a font ID. If the font ID corresponds to the default font for a particular family, `#f` is returned.

- (send *a-font-name-directory* `get-face-name` *font-id*) ⇒ string or `#f`
  font-id : exact integer

get-family

Gets the family for a font ID. See `font%` for information about font families.

- (send *a-font-name-directory* `get-family` *font-id*) ⇒ symbol in `'(default decorative roman script swiss modern symbol system)`
  font-id : exact integer

get-font-id

Gets the font ID for a face name paired with a default family. If the mapping for the given pair is not already initialized, `0` is returned. See also `find-or-create-font-id`.

Font ID are useful only as mapping indices for `the-font-name-directory`.

- (send *a-font-name-directory* `get-font-id` *name family*) ⇒ exact integer
  name : string
  family : symbol   in   `'(default decorative roman script swiss modern symbol system)`

`get-post-script-name`

Gets a PostScript font name for a font ID, weight, and style combination. The PostScript font name is used both for the font name in PostScript output, and as the Adobe(tm) Font Manager file name (suffixed with `".afm"` and prefixed with the Adobe(tm) Font Manager path determined by `get-afm-path`).

- (send *a-font-name-directory* `get-post-script-name` *font-id weight style*) $\Rightarrow$ string or #f
  font-id : exact integer
  weight : symbol in '(normal bold light)
  style : symbol in '(normal italic slant)

See `font%` for information about *weight* and *style*.

`get-screen-name`

Gets a platform-dependent screen font name (used for drawing to a canvas's `dc<%>`, a `bitmap-dc%`, or a `printer-dc%`) for a font ID, weight, and style combination.

- (send *a-font-name-directory* `get-screen-name` *font-id weight style*) $\Rightarrow$ string or #f
  font-id : exact integer
  weight : symbol in '(normal bold light)
  style : symbol in '(normal italic slant)

See `font%` for information about *weight* and *style*.

`set-post-script-name`

Sets a PostScript font name for a font ID, weight, and style combination. See also `get-post-script-name`.

- (send *a-font-name-directory* `set-post-script-name` *font-id weight style name*) $\Rightarrow$ void
  font-id : exact integer
  weight : symbol in '(normal bold light)
  style : symbol in '(normal italic slant)
  name : string

See `font%` for information about *weight* and *style*.

`set-screen-name`

Sets a platform-dependent screen font name (used for drawing to a canvas's `dc<%>`, a `bitmap-dc%`, or a `printer-dc%`) for a font ID, weight, and style combination.

Under X, if the screen name contains "%d," it is replaced by the size of the font (point size times 10) to obtain the full screen font name.

- (send *a-font-name-directory* `set-screen-name` *font-id weight style name*) $\Rightarrow$ void
  font-id : exact integer
  weight : symbol in '(normal bold light)
  style : symbol in '(normal italic slant)
  name : string

See `font%` for information about *weight* and *style*.

6.12   `pen%`

A pen is a drawing tool with a color, width, and style. A pen draws lines and outlines, such as the outline of a rectangle. On a monochrome display, all non-white pens are drawn as black.

In addition to its color, width, and style, a pen can have a stipple bitmap that is a 8 x 8 monochrome bitmap. Painting with a stipple pen is similar to calling `draw-bitmap` with the stipple bitmap in region painted by the pen.

A pen's style is one of the following:

- `'transparent` — Draws with no effect (on the outline of the drawn shape).

- `'solid` — Draws using the pen's color. If a (monochrome) stipple is installed into the pen, black pixels from the stipple are transferred to the destination using the brush's color, and white pixels from the stipple are not transferred.

- `'xor` — The pen's color or colored stipple is *xor*-ed with existing destination pixel values. The `'xor` mapping is unspecified for arbitrary color combinations, but the mapping provides two guarantees:
  - Black-and-white drawing to a color or monochrome destination always works as expected: black xor white = black, white xor black = black, black xor black = white, and white xor white = white.
  - Performing the same drawing operation twice in a row with `'xor` is equivalent to a no-op.

- The following special pen modes use the pen's color and only apply when a stipple is not installed:
  - `'dot`
  - `'long-dash`
  - `'short-dash`
  - `'dot-dash`
  - `'xor-dot`
  - `'xor-long-dash`
  - `'xor-short-dash`
  - `'xor-dot-dash`

To avoid creating multiple pens with the same characteristics, use the global `pen-list%` object `the-pen-list`.

A pen of size `0` uses the minimum line size for the destination drawing context. In (unscaled) screens and bitmaps, this behaves the nearly same as a pen of size `1`. In a `post-script-dc%`, a pen of size `0` draws a line thinner than a pen of size `1`. If the pen's width is not an integer, then the width is truncated to an integer (even before scaling) when *not* drawing to a `post-script-dc%`.

- `(make-object pen%)` ⇒ `pen%` object

- `(make-object pen% ` *color width style*`)` ⇒ `pen%` object
    color : `color%` object
    width : real number in [0, 255]
    style : symbol   in   `'(transparent solid xor dot long-dash short-dash dot-dash`
          `xor-dot xor-long-dash xor-short-dash xor-dot-dash)`
  Creates a pen using a color object.

- `(make-object pen% ` *color-name width style*`)` ⇒ `pen%` object
    color-name : string

> > *width* : real number in [0, 255]
> > *style* : symbol   in   '(transparent solid xor dot long-dash short-dash dot-dash
> >   xor-dot xor-long-dash xor-short-dash xor-dot-dash)
>
> Creates a pen using a color name; a color is found for the name through the global `color-database<%>`
> object `the-color-database`. If the color name is not known, the pen is initialized to black.

get-cap

Returns the pen cap style (X, Windows). The default is '`round`.

- (send *a-pen* get-cap) ⇒ symbol in '(round projecting butt)

get-color

Returns the pen's color object.

- (send *a-pen* get-color) ⇒ color% object

get-join

Returns the pen join style (X, Windows). The default is '`round`.

- (send *a-pen* get-join) ⇒ symbol in '(round bevel miter)

get-stipple

Gets the current stipple bitmap, or return `#f` is no stipple bitmap is being used.

- (send *a-pen* get-stipple) ⇒ bitmap% object or #f

get-style

Returns the pen style. See `pen%` for information about possible styles.

- (send *a-pen* get-style) ⇒ symbol in '(transparent solid xor dot long-dash short-dash
  dot-dash xor-dot xor-long-dash xor-short-dash xor-dot-dash)

get-width

Returns the pen width.

- (send *a-pen* get-width) ⇒ real number in [0, 255]

set-cap

Sets the pen cap style (X, Windows). See `get-cap` for information about cap styles.

A pen cannot be modified if it was obtained from a `pen-list%` or while it is selected into a drawing context.

- (send *a-pen* `set-cap` *cap-style*) ⇒ void
    cap-style : symbol in '(round projecting butt)

set-color

Sets the pen color.

A pen cannot be modified if it was obtained from a `pen-list%` or while it is selected into a drawing context.

- (send *a-pen* `set-color` *color*) ⇒ void
    color : `color%` object

    Sets the color to match the given color.

- (send *a-pen* `set-color` *color-name*) ⇒ void
    color-name : string

    Sets the pen color by looking up a color name in the global `color-database<%>` object `the-color-database`. The pen is not changed if the color is unknown.

- (send *a-pen* `set-color` *red green blue*) ⇒ void
    red : exact integer in [0, 255]
    green : exact integer in [0, 255]
    blue : exact integer in [0, 255]

    Sets the pen color to specific RGB values.

set-join

Sets the pen join style (X, Windows). See `get-join` for information about join styles.

A pen cannot be modified if it was obtained from a `pen-list%` or while it is selected into a drawing context.

- (send *a-pen* `set-join` *join-style*) ⇒ void
    join-style : symbol in '(round bevel miter)

set-stipple

Sets the pen stipple bitmap, which must be an 8 x 8 monochrome bitmap or `#f`, which turns off the stipple bitmap.

A bitmap cannot be used as a stipple if it is selected into a `bitmap-dc%` object; if the given bitmap is selected into a `bitmap-dc%` object, an `exn:application:mismatch` exception is raised. A pen cannot be modified if it was obtained from a `pen-list%` or while it is selected into a drawing context.

- (send *a-pen* `set-stipple` *stipple*) ⇒ void
    stipple : `bitmap%` object or `#f`

set-style

Sets the pen style. See `pen%` for information about the possible styles.

A pen cannot be modified if it was obtained from a `pen-list%` or while it is selected into a drawing context.

- (send *a-pen* `set-style` *style*) ⇒ void
    style : symbol   in   '(transparent solid xor dot long-dash short-dash dot-dash
           xor-dot xor-long-dash xor-short-dash xor-dot-dash)

`set-width`

Sets the pen width.

A pen cannot be modified if it was obtained from a `pen-list%` or while it is selected into a drawing context.

- (send *a-pen* `set-width` *width*) ⇒ void
    width : real number in [0, 255]

6.13   `pen-list%`

A `pen-list%` object maintains a list of `pen%` objects to avoid repeatedly creating pen objects. A `pen%` object in a pen list cannot be mutated.

A global pen list `the-pen-list` is created automatically.

- (`make-object pen-list%`) ⇒ `pen-list%` object

    Creates an empty pen list.

`find-or-create-pen`

Finds a pen of the given specification, or creates one and adds it to the list.

- (send *a-pen-list* `find-or-create-pen` *color* *width* *style*) ⇒ `pen%` object
    color : `color%` object
    width : real number in [0, 255]
    style : symbol   in   '(transparent solid xor dot long-dash short-dash dot-dash
           xor-dot xor-long-dash xor-short-dash xor-dot-dash)

    See `pen%`.

- (send *a-pen-list* `find-or-create-pen` *color-name* *width* *style*) ⇒ `pen%` object or `#f`
    color-name : string
    width : real number in [0, 255]
    style : symbol   in   '(transparent solid xor dot long-dash short-dash dot-dash
           xor-dot xor-long-dash xor-short-dash xor-dot-dash)

    See `pen%`.

    The return value is `#f` when no color matching *color-name* can be found in the color database.

6.14   `point%`

A `point%` is used for certain drawing commands. It excapsulates two real numbers, x and y.

- (`make-object point%`) ⇒ `point%` object

    Creates a point with 0 x and y values.

- (make-object point% *x* *y*) ⇒ point% object
  x : real number
  y : real number

    Creates a point.

get-x

Gets the point x-value.

- (send *a-point* get-x) ⇒ real number

get-y

Gets the point y-value.

- (send *a-point* get-y) ⇒ real number

set-x

Sets the point x-value.

- (send *a-point* set-x *x*) ⇒ void
  x : real number

set-y

Sets the point y-value.

- (send *a-point* set-y *y*) ⇒ void
  y : real number

6.15   `post-script-dc%`

Implements: `dc<%>`

A `post-script-dc%` object is a PostScript device context, that can write PostScript files on any platform. See also `ps-setup%`.

Be sure to use the following methods to start/end drawing:

- `start-doc`
- `start-page`
- `end-page`
- `end-doc`

See also `printer-dc%`.

- (make-object post-script-dc% *interactive?* *parent*) ⇒ `post-script-dc%` object
  interactive? = `#t` : boolean
  parent = `#f` : `frame%` or `dialog%` object or `#f`

If *interative?* is true, the user is given a dialog for setting printing parameters (see `get-ps-setup-from-user`); the resulting configuration is installed as the current configuration). If the user chooses to print to a file (the only possibility under Windows and MacOS), another dialog is given to select the filename. If the user hits cancel in either of these dialogs, then `ok?` returns `#f`.

If *parent* is not `#f`, it is used as the parent window of the configuration dialog.

If *interative?* is `#f`, then the settings returned by `current-ps-setup` are used. A file dialog is still presented to the user if the `get-file` method returns `#f`, and the user may hit cancel in that case so that `ok?` returns `#f`.

See also `ps-setup%` and `current-ps-setup`. The settings for a particular `post-script-dc%` object are fixed to the values in the current configuration when the object is created (after the user has interactively adjusted them when *interactive?* is true).

6.16 `printer-dc%`

Implements: `dc<%>`

A `printer-dc%` object is a Windows or MacOS printer device context. The class cannot be instantiated under X (an `exn:misc:unsupported` exception is raised).

Be sure to use the following methods to start/end drawing:

- `start-doc`

- `start-page`

- `end-page`

- `end-doc`

See also `post-script-dc%`.

When a `printer-dc%` object is created, the user gets platform-specific modal dialogs for configuring the output.

- (make-object printer-dc% *parent*) ⇒ `printer-dc%` object
  parent = `#f` : `frame%` or `dialog%` object or `#f`

If *parent* is not `#f`, it is used as the parent window of the configuration dialog.

6.17 `ps-setup%`

A `ps-setup%` object contains configuration information for producing PostScript files using a `post-script-dc%` object.

When a `post-script-dc%` object is created, its configuration is determined by the `current-ps-setup` parameter's `ps-setup%` value. After a `post-script-dc%` object is created, it is unaffected by changes to the `current-ps-setup` parameter or mutations to the `ps-setup%` object.

- (make-object ps-setup%) ⇒ ps-setup% object

  Creates a new `ps-setup%` object with the (platform-specific) default configuration.

copy-from

Copies the settings from the given `ps-setup%` object to this one.

- (send *a-ps-setup* copy-from *source*) ⇒ void
    source : ps-setup% object

get-afm-path

- (send *a-ps-setup* get-afm-path) ⇒ string or #f

  Returns the current Adobe(tm) Font Manager directory pathname, used for getting font size information when generating PostScript files. If the directory is unknown, #f is returned.

  If Adobe(tm) Font Manager files cannot be found when a PostScript file is being generated, then text sizes will be cacluated incorrectly and will likely be drawn with an incorrect position.

get-command

- (send *a-ps-setup* get-command) ⇒ string

  Gets the printer command used to print a file in X. The default is usually `"lpr"`. This value is not used by other platforms.

get-editor-margin

- (send *a-ps-setup* get-editor-margin *h-margin* *v-margin*) ⇒ void
    h-margin : boxed exact non-negative integer
    v-margin : boxed exact non-negative integer

  Returns the current settings for horizontal and vertical margins when printing an `editor<%>`. See also `set-editor-margin`.

get-file

- (send *a-ps-setup* get-file) ⇒ string or #f

  Gets the PostScript output filename. A #f value (the default) indicates that the user should be prompted for a filename when a `post-script-dc%` object is created.

get-level-2

- (send *a-ps-setup* get-level-2) ⇒ boolean

  Reports whether Level 2 commands are output in PostScript files.

  Currently, Level 2 commands are only needed to include color bitmap images in PostScript output (drawn with `draw-bitmap`), or bitmap pen and brush stipples. When Level 2 commands are disabled, bitmaps are converted to grayscale images and stipples are not supported.

`get-margin`

- (send *a-ps-setup* `get-margin` *h-margin v-margin*) ⇒ void
    h-margin : boxed non-negative real number
    v-margin : boxed non-negative real number

Returns the current settings for horizontal and vertical PostScript margins. See also `set-margin`.

`get-mode`

- (send *a-ps-setup* `get-mode`) ⇒ symbol in '(`preview file printer`)

Gets the printing mode that determines where output is sent: '`preview`, '`file`, or '`printer`. The default for X is '`preview`. The value Windows and MacOS is always '`file`.

`get-options`

- (send *a-ps-setup* `get-options`) ⇒ string

Gets the additional options for the print command (e.g. specific printer). The default is `""`.

`get-orientation`

- (send *a-ps-setup* `get-orientation`) ⇒ symbol in '(`portrait landscape`)

Gets the orientation: '`portrait` or '`landscape`. The default is '`portrait`.

Landscaped orientation affects the size of the drawing area as reported by `get-size`: the horizontal and vertical sizes determined by the selected paper type are transposed and then scaled.

`get-paper-name`

- (send *a-ps-setup* `get-paper-name`) ⇒ string

Returns the name of the current paper type: `"A4 210 x 297 mm"`, `"A3 297 x 420 mm"`, `"Letter 8 1/2 x 11 in"`, or `"Legal 8 1/2 x 14 in"`. The default is `"Letter 8 1/2 x 11 in"`. The paper name determines the size of the drawing area as reported by `get-size` (along with landscape transformations from `get-orientation` and/or the scaling factors of `get-scaling`).

`get-preview-command`

- (send *a-ps-setup* `get-preview-command`) ⇒ string

Gets the command used to view a PostScript file for X. The default is usually `"ghostview"`. This value is not used by other platforms.

`get-scaling`

- (send *a-ps-setup* `get-scaling` *x y*) ⇒ void
    x : boxed non-negative real number
    y : boxed non-negative real number

Gets the scaling factor for PostScript output. The x box is filled with the horizontal scaling factor. The y box is filled with the vertical scaling factor. The default is `1.0` by `1.0`.

This scale is in addition to a scale that can be set by `set-scale` in a `post-script-dc%` context. The size reported by `get-size` is the size of the selected paper type (transposed for landscaped mode) divided by this scale.

`get-translation`

- (send *a-ps-setup* `get-translation` *x y*) ⇒ void
    - *x* : boxed real number
    - *y* : boxed real number

Gets the translation (from the bottom left corner) for PostScript output. The *x* box is filled with the horizontal offset. The *y* box is filled with the vertical offset. The default is `0.0` and `0.0`.

The translation is not scaled by the numbers returned from `get-scaling` and the translation does not affect the size of the drawing area.

`set-afm-path`

- (send *a-ps-setup* `set-afm-path` *path*) ⇒ void
    - *path* : string or `#f`

Sets the current Adobe(tm) Font Manager directory pathname. See `get-afm-path`.

`set-command`

- (send *a-ps-setup* `set-command` *command*) ⇒ void
    - *command* : string

Sets the printer command used to print a file under X. See `get-command`.

`set-editor-margin`

- (send *a-ps-setup* `set-editor-margin` *h v*) ⇒ void
    - *h* : exact non-negative integer
    - *v* : exact non-negative integer

Sets the horizontal and vertical margins used when when printing an editor with the `print` method. These margins are always used for printing, whether the drawing destination is a `post-script-dc%` or `printer-dc%`. The margins are in the units of the destination `printer-dc%` or `post-script-dc%`. In the case of `post-script-dc%` printing, the editor margin is in addition to the PostScript margin that is determined by `set-margin`.

`set-file`

- (send *a-ps-setup* `set-file` *filename*) ⇒ void
    - *filename* : string or `#f`

Sets the PostScript output filename. See `get-file`.

`set-level-2`

- (send *a-ps-setup* `set-level-2` *on?*) ⇒ void
    - *on?* : boolean

Sets whether Level 2 commands are output in PostScript files. See `get-level-2`.

`set-margin`

- (send *a-ps-setup* `set-margin` *h v*) ⇒ void
    - *h* : non-negative real number

v : non-negative real number

Sets the horizontal and vertical PostScript margins. When drawing to a `post-script-dc%`, the page size reported by `get-size` subtracts these margins from the normal page area (before taking into account scaling affects). In addition, drawing into the `post-script-dc%` produces PostScript output that is offset by the margins.

When using the output of a `post-script-dc%` as Encapsulated PostScript, the margin values are effectively irrelevant. Changing the margins moves the PostScript image in absolute coordinates, but it also moves the bounding box.

The margins are in unscaled `post-script-dc%` units, which are points. The default margins are 16 points.

set-mode

- (send *a-ps-setup* `set-mode` *mode*) $\Rightarrow$ void
    mode : symbol in '(preview file printer)

Sets the printing mode controlling where output is sent. See `get-mode`.

Under Windows and MacOS, if 'preview or 'printer is provided, an `exn:application:mismatch` exception is raised.

set-options

- (send *a-ps-setup* `set-options` *options*) $\Rightarrow$ void
    options : string

Sets the additional options for the print command (e.g. specific printer) under X. See `get-options`.

set-orientation

- (send *a-ps-setup* `set-orientation` *orientation*) $\Rightarrow$ void
    orientation : symbol in '(portrait landscape)

Sets the orientation. See `get-orientation`.

set-paper-name

- (send *a-ps-setup* `set-paper-name` *type*) $\Rightarrow$ void
    type : string

Sets the name of the current paper type. See `get-paper-name`.

set-preview-command

- (send *a-ps-setup* `set-preview-command` *command*) $\Rightarrow$ void
    command : string

Sets the command used to view a PostScript file under X. See `get-preview-command`.

set-scaling

- (send *a-ps-setup* `set-scaling` *x* *y*) $\Rightarrow$ void
    x : non-negative real number
    y : non-negative real number

Sets the scaling factor for PostScript output. See `get-scaling`.

set-translation

- (send *a-ps-setup* `set-translation` *x* *y*) ⇒ void
    x : real number
    y : real number

Sets the translation (from the bottom left corner) for PostScript output. See `get-translation`.

6.18   `region%`

A `region%` object specifies a portion of a drawing area (possibly discontinuous) for clipping drawing operations.

Each `region%` object is associated to a particular `dc<%>` object, specified when the region is created. A region can only be used with its associated `dc<%>` object, and changing the origin or scale of a drawing context invalidates its associated regions. (The region can still be used after the origin or scale is chaged, but the results are platform- and device-dependent.)

See also `set-clipping-region` in `dc<%>` and `get-clipping-region` in `dc<%>`.

- (`make-object` `region%` *dc*) ⇒ `region%` object
    dc : `dc<%>` object

Creates an empty region.

get-bounding-box

Returns a rectangle that encloses the region. The return values are the top, left, width, and height of the rectangle.

- (send *a-region* `get-bounding-box`) ⇒ four real numbers

get-dc

Returns the region's drawing context.

- (send *a-region* `get-dc`) ⇒ `dc<%>` object

intersect

Sets the region to the intersection of itself with the given region.

This region's DC and given region's DC must be the same.

- (send *a-region* `intersect` *rgn*) ⇒ void
    rgn : `region%` object

`is-empty?`

Returns `#t` if the region is empty, `#f` otherwise.

- (send *a-region* `is-empty?`) $\Rightarrow$ boolean

`set-arc`

Sets the region to the interior of the specified wedge.

See also `draw-ellipse` in `dc<%>`.

- (send *a-region* `set-arc` *x y width height start-radians end-radians*) $\Rightarrow$ void
  x : real number
  y : real number
  width : non-negative real number
  height : non-negative real number
  start-radians : real number
  end-radians : real number

`set-ellipse`

Sets the region to the interior of the specified ellipse.

See also `draw-ellipse` in `dc<%>`.

- (send *a-region* `set-ellipse` *x y width height*) $\Rightarrow$ void
  x : real number
  y : real number
  width : non-negative real number
  height : non-negative real number

  Restrictions on the magnitude of drawing coordinates are described with `dc<%>`.

`set-polygon`

Sets the region to the interior of the specified polygon.

See also `draw-polygon` in `dc<%>`.

- (send *a-region* `set-polygon` *points xoffset yoffset fill-style*) $\Rightarrow$ void
  points : list of `point%` objects
  xoffset = 0 : real number
  yoffset = 0 : real number
  fill-style = `'odd-even` : symbol in `'(odd-even winding)`

`set-rectangle`

Sets the region to the interior of the specified rectangle.

See also `draw-rectangle` in `dc<%>`.

- (send *a-region* `set-rectangle` *x y width height*) ⇒ void
    x : real number
    y : real number
    width : non-negative real number
    height : non-negative real number

Restrictions on the magnitude of drawing coordinates are described with `dc<%>`.

set-rounded-rectangle

Sets the region to the interior of the specified rounded rectangle.

See also `draw-rounded-rectangle` in `dc<%>`.

- (send *a-region* `set-rounded-rectangle` *x y width height radius*) ⇒ void
    x : real number
    y : real number
    width : non-negative real number
    height : non-negative real number
    radius = 20 : real number

Restrictions on the magnitude of drawing coordinates are described with `dc<%>`.

subtract

Sets the region to the subtraction of itself minus the given region. In other words, a point is removed from the region if it is included in the given region. (The given region may contain points that are not in the current region; such points are ignored.)

This region's DC and given region's DC must be the same.

- (send *a-region* `subtract` *rgn*) ⇒ void
    rgn : `region%` object

union

Sets the region to the union of itself with the given region.

This region's DC and given region's DC must be the same.

- (send *a-region* `union` *rgn*) ⇒ void
    rgn : `region%` object

7.   Drawing Procedures

7.1   Global Graphics and Screen

`current-ps-setup`

Sets the object that contains the current PostScript configuration settings.

- (`current-ps-setup`) ⇒ `ps-setup%` object

  Returns the current PostScript configuration object.

- (`current-ps-setup` *ps-setup*) ⇒ `void`
    ps-setup : `ps-setup%` object

  Sets the current PostScript configuration object.

`flush-display`

Under X, flushes pending display messages such that the user's display reflects the actual state of the windows. Under Windows and MacOS, the procedure has no effect.

- (`flush-display`) ⇒ `void`

`get-display-depth`

- (`get-display-depth`) ⇒ `exact non-negative integer`

  Returns the depth of the main display (a value of 1 denotes a monochrome display).

`get-display-size`

Gets the physical size of the display in pixels. Under Windows, this size does not include the task bar. Under MacOS, this size does not include the menu bar.

- (`get-display-size`) ⇒ `two exact non-negative integers`

  Returns the screen's width and height.

`get-face-list`

Returns a list of font face names available on the current system.

- (`get-face-list`) ⇒ `list of strings`

`get-family-builtin-face`

Returns the built-in default face mapping for a particular font family. The built-in default can be overridden via resources, as described in the **FONTS** file that is included in the **notes** directory of the MrEd distribution.

- (get-family-builtin-face *family*) ⇒ string
  family : symbol in '(default decorative roman script swiss modern symbol system)

  See font% for information about *family*.

`is-color-display?`

- (is-color-display?) ⇒ boolean

  Returns #t if the main display has color, #f otherwise.

`register-collecting-blit`

Registers a blit to occur when garbage collection starts or ends.

- (register-collecting-blit *canvas x y w h on off on-x on-y off-x off-y*) ⇒ void
  canvas : canvas% object
  x : real number
  y : real number
  w : non-negative real number
  h : non-negative real number
  on : bitmap% object
  off : bitmap% object
  on-x = 0 : real number
  on-y = 0 : real number
  off-x = 0 : real number
  off-y = 0 : real number

  When garbage collection starts, (send (send *canvas* get-dc) draw-bitmap-section *on on-x on-y x y w h*) is called. When garbage collection ends, (send (send *canvas* get-dc) draw-bitmap-section *off off-x off-y x y w h*) is called.

  The *canvas* is registered weakly, so it will be automatically unregistered if the canvas becomes invisible and inaccessible. Multiple registrations can be installed for the same canvas.

  See also unregister-collecting-blit.

`the-brush-list`

See brush-list%.

- the-brush-list ⇒ brush-list% object
  Initial value : empty brush list

`the-color-database`

See color-database<%>.

- `the-color-database` ⇒ `color-database<%>` object
  Initial value : basic color database

the-font-list

See `font-list%`.

- `the-font-list` ⇒ `font-list%` object
  Initial value : empty font list

the-font-name-directory

See `font-name-directory<%>`.

- `the-font-name-directory` ⇒ `font-name-directory<%>` object
  Initial value : the font name directory

the-pen-list

See `pen-list%`.

- `the-pen-list` ⇒ `pen-list%` object
  Initial value : empty pen list

unregister-collecting-blit

Unregisters a blit request installed with See also `register-collecting-blit`.

- (`unregister-collecting-blit` *canvas*) ⇒ void
  canvas : `canvas%` object

  Unregsiters all blits for *canvas*.

Part III

Editor Toolbox

8.   Editor Toolbox

The editor toolbox provides a foundation for two common kinds of applications:

1. **Programs that need a sophisticated text editor** — The simple text field control is inadequate for text-intensive applications. Many programs need editors that can handle multiple fonts and non-text items.

2. **Programs that need a canvas with dragable objects** — The drawing toolbox provides a generic drawing surface for plotting lines and boxes, but many applications need an interactive canvas, where the user can drag and resize individual objects.

Both kinds of applications need an extensible editor that can handle text, images, programmer-defined items, and even embedded editors. The difference between them is the layout of items. MrEd therefore provides two kinds of editors via two classes:

- `text%` — in a **text editor**, items are automatically positioned in a paragraph flow.

- `pasteboard%` — in a **pasteboard editor**, items are explicitly positioned and dragable.

MrEd's editor architecture addresses the full range of real-world issues for an editor—including cut-and-paste, extensible file formats, and layered text styles—while supporting a high level of extensibility. Unfortunately, the system is fairly complex as a result,[1] and using the editor classes effectively requires a solid understanding of the structure and terminology of the editor toolbox. Nevertheless, enough applications fit one (or both) of the descriptions above to justify the depth and complexity of the toolbox and the learning investment required to use it.

A brief example illustrates how MrEd editors work. To start, an editor needs an `editor-canvas%` to display its contents. Then, we can create a text editor an install it into the canvas:

```
(define f (make-object frame% "Simple Edit" #f 200 200))
(define c (make-object editor-canvas% f))
(define t (make-object text%))
(send c set-editor t)
(send f show #t)
```

At this point, the editor is fully functional: the user can type text into the editor, but no cut-and-paste operations are available. We can support all of the standard operations on an editor via the menu bar:

```
(define mb (make-object menu-bar% f))
(define m-edit (make-object menu% "Edit" mb))
(define m-font (make-object menu% "Font" mb))
(append-editor-operation-menu-items m-edit)
(append-editor-font-menu-items m-font)
```

---

[1]Nearly half of this manual is dedicated to documenting the editor classes.

146

Now, the standard cut and paste operations work, and the user can even set font styles. The user can also insert an embedded editor by selecting Insert Text from the Edit menu; after selecting the menu item, a box appears in the editor with the caret inside. Typing with the caret in the box stretches the box as text is added, and font operations apply wherever the caret is active. Text on the outside of the box is rearranged as the box changes sizes. Note that the box itself can be copied and pasted.

The content of an editor is made up of **snips**. An embedded editor is a single snip from the embedding editor's point-of-view. To encode immediate text, a snip can be a single character, but more often a snip is a sequence of adjacent characters on the same line. The `find-snip` method extracts a snip from a text editor:

```
(send t find-snip 0 'after)
```

The above expression returns the first snip in the editor, which may be a string snip (for immediate text) or an editor snip (for an embedded editor).

An editor is not permanently attached to any display. We can take the text editor out of our canvas and put a pasteboard editor in the canvas, instead:

```
(define pb (make-object pasteboard%))
(send c set-editor pb)
```

With the pasteboard editor installed, the user can no longer type characters directly into the editor (because a pasteboard does not support directly entered text). However, the user can cut text from elsewhere and paste it into pasteboard, or select one of the Insert menu items in the Edit menu. Snips are clearly identifiable in a pasteboard editor (unlike a text editor) because each snip is separately dragable.

We can insert the old text editor (which we recently removed from the canvas) as an embedded editor in the pasteboard by explicitly creating an editor snip:

```
(define s (make-object editor-snip% t)) ; t is the old text editor
(send pb insert s)
```

An individual snip cannot be inserted into different editors at the same time, or inserted multiple times in the same editor:

```
(send pb insert s) ; no effect
```

However, we can make a deep copy of the snip and insert the copy into the pasteboard:

```
(send pb insert (send s copy))
```

Applications that use the editor classes typically derive new versions of the `text%` and `pasteboard%` classes. For example, to implement an append-only editor (which allows insertions only at the end and never allows deletions), derive a new class from `text%` and override the `can-insert?` and `can-delete?` methods:

```
(define append-only-text%
  (class text% args
    (inherit last-position)
    (override
     [can-insert? (lambda (s l) (= s (last-position)))]
     [can-delete? (lambda (s l) #f)])
    (sequence (apply super-init args))))
```

8.1   Editor Structure and Terminology

MrEd supports extensible and nestable editors by decomposing an editor assembly into three functional parts:

- The **editor** itself stores the state of the text or pasteboard and handles most events and editing operations. The `editor<%>` interface defines the core editor functionality, but editors are created as instances of `text%` or `pasteboard%`.

- A **snip** is a segment of information within the editor. Each snip can contain a sequence of characters, a picture, or an interactive object (such as an embedded editor). In a text editor, snips are constrained to fit on a single line and generally contain data of a single type. The `snip%` class implements a basic snip. Other snip classes include `string-snip%` for managing text, `image-snip%` for managing pictures, and `editor-snip%` for managing embedded editors.

- A **display** presents the editor on the screen. The display lets the user scroll around an editor or change editors. Most displays are instances of the `editor-canvas%` class, but the `editor-snip%` class also acts as a display for embedded editors.

These three parts are illustrated by a simple word processor. The editor corresponds to the text document. The editor object receives keyboard and mouse commands for editing the text. The text itself is distributed among snips. Each character could be a separate snip, or multiple characters on a single line could be grouped together into a snip. The display roughly corresponds to the window in which the text is displayed. While the editor manages the arrangement of the text as it is displayed into a window, the display determines which window to draw into and which part of the editor to display.

Each selectable entity in an editor is an **item**. In a pasteboard, all selection and dragging operations work on snips, so there is a one-to-one correspondence between snips and items. In an editor, one snip contains one or more consecutive items, and every item belongs to some snip. For example, in a simple text editor, each character is an item, but multiple adjacent characters may be grouped into a single snip.

Each place where the insertion point can appear in a text editor is a **text position**. A text editor with n items contains $n + 1$ text positions: one position before each item, and one position after the last item.

The order of snips within a pasteboard determines each snip's drawing plane. When two snips overlap within the pasteboard, the snip that is earlier in the order is in front of the other snip (i.e., the former is drawn after the latter, such that the former snip may cover part of the latter snip).

When an editor is drawn into a display, each snip and text position has a **graphic location**. The location of a position or snip is specified in coordinates relative to the top-left corner of the editor. Locations in an editor are only meaningful when the editor is displayed.

8.1.1   Administrators

Two extra layers of administration manage the display-editor and editor-snip connections. An editor never communicates directly with a display; instead, it always communicates with a **editor administrator**, an instance of the `editor-admin%` class, which relays information to the display. Similarly, a snip communicates with a **snip administrator**, an instance of the `snip-admin%` class.

The administrative layers make the editor hierarchy flexible without forcing every part of an editor assembly to contain the functionality of several parts. For example, a text editor can be a single item within another editor; without administrators, the `text%` class would also have to contain all the functionality of a display (for the containing editor) and a snip (for the embedded editor). Using administrators, an editor class can

serve as both a containing and an embedded editor without directly implementing the display and snip functionality.

A snip belongs to at most one editor via a single administrator. A editor also has only one administrator at a time. However, the administrator that connects the an editor to the standard display (i.e., an editor canvas) can work with other such administrators. In particular, the administrator of a `editor-canvas%` (each one has its own administrator) can work with other `editor-canvas%` administrators, allowing an editor to be displayed in multiple `editor-canvas%` windows at the same time.

When an editor is displayed by multiple canvases, one of the canvases' administrators is used as the editor's primary administrator. To handle user and update events for other canvases, the editor's administrator is temporarily changed and then restored through the editor's `set-admin` method. The return value of the editor's `get-admin` method thus depends on the context of the call.

8.1.2 Styles

A **style**, an instance of the `style<%>` interface, parameterizes high-level display information that is common to all snip classes. This includes the font, color, and alignment for drawing the item. A single style is attached to each snip.

Styles are hierarchical: each style is defined in terms of another style. There is a single **root style**, named `"Basic"`, from which all other styles in an editor are derived. The difference between a base style and each of its derived style is encoded in a **style delta** (or simply **delta**). A delta encodes changes such as

- change the font family to X;

- enlarge the font by adding Y to the point size;

- toggle the boldness of the font; or

- change everything to match the style description Z.

Style objects are never created separately; rather, they are always be created through a **style list**, an instance of the `style-list%` class. A style list manages the styles, servicing external requests to find a particular style, as well as the hierarchical relationship between the styles. A global style list is available (`the-style-list`), but new style lists can be created for managing separate style hierarchies. For example, each editor will typically have its own style list.

Each new style is defined in one of two ways:

- A **derived style** is defined in terms of a base style and a delta. Every style (except for the root style) has a base style, even if it does not depend on the base style in any way (i.e., the delta describes a fixed style rather than extensions to an existing style).[2]

- A **join style** is defined in terms of two other styles: a base style and a **shift style**. The meaning of a join style is determined by reinterpreting the shift style; in the reinterpretation, the base style is used as the *root* style for the shift style.[3]

Usually, when text is inserted into a buffer, a it inherits the style of the preceeding snip. If text is inserted into an empty editor, the new snip is usually assigned a style called `"Standard"`. By default, the `"Standard"` style is unmodified from the root style.

---

[2] This is the usual kind of style inheritance, as found in word processors such as Microsoft Word.

[3] This is analogous to multi-level styles, like the paragraph and character styles in FrameMaker. In this analogy, the paragraph style is the base style, and the character style is the shift style. However, FrameMaker allows only those two levels; with join styles support any number of levels.

The exception to the above is when `change-style` in `text%` is called with the current selection position (when the selection is a position and not a range). In that case, the style is remembered, and if the next buffer-modifying action is a text insertion, the inserted text gets the remembered style.

8.2 File Format

Editor data can be read and written using `editor-stream-in%` and `editor-stream-out%` objects.

Editor information can only be read from or written to one stream at a time. To write one or more editors to a stream, first call the function `write-editor-global-header` to write initialization data into an output stream. When all editors are written to the stream, call `write-editor-global-footer`. Similarly, reading editors from a stream is initialized with `read-editor-global-header` and finalized with `read-editor-global-footer`.

The editor file data format can be embedded within another file, and it can be extended with new kinds of data. The editor file format can be extended in two ways: with snip- or content-specific data, and with editor-specific global data. These are descibed in the remainder of this section.

8.2.1   Encoding Snips

The generalized notion of a snip allows new snip types to be defined and immediately used in any editor class. Also, when two applications support the same kinds of snips, snip data can easily be cut and pasted between them, and the same data files will be readable by each program. This interoperability is due to a consistent encoding mechanism that is built into the snip system.

Graceful and extensible encoding of snips requires that two issues are addressed:

- In order to convert a snip from an encoded representation (e.g., as bytes in a file) to a memory object, a decoding function must be provided for each type of snip. Furthermore, a list of such decoders must be available to the high-level decoding process. This decoding mapping is defined by associating a **snip class** object to every snip. A snip class is an instance of the `snip-class%` class.

- Some editors may require additional information to be stored about a snip; this information is orthogonal to the type-specific information stored by the snip itself. For example, a pasteboard needs to remember a snip's position, while a text editor does not need this information. If data is being cut and pasted from one pasteboard to another, then information about relative positions needs to be maintained, but this information should not inhibit pasting into an editor. Extra data is associated with a snip through **editor data** objects, instances of the `editor-data%` class; decoding requires that each editor data object has a **editor data class**, an instance of the `editor-data-class%` class.

Snip classes, snip data, and snip data classes solve problems related to encoding and decoding snips. In an application that has no need for saving files or cut-and-paste, these issues can be safely ignored.

Snip Classes   Each snip can be associated to a **snip class**. This "class" is not a class description in the programmer's language; it is an object which provides a way to create new snips of the appropriate type from an encoded snip specification. All snip class objects should be added to the global **snip class list**, returned by `get-the-snip-class-list`.

When a snip is encoded, the snip's class name is associated with the encoding; when the snip needs to be decoded, then the snip class list is searched by name to find the snip's class. The snip class will then provide a decoding function that can create a new snip from the encoding.

Editor Data While a snip belongs to a editor, the editor may store extra information about a snip in some specialized way. When the snip is to be encoded, this extra information needs to be put into a **editor data** object so that the extra information can be encoded as well. In a text editor, extra information can be associated with ranges of items, as well as snips.

Just as a snip must be associated with a snip class to be decoded (see section 8.2.1 (page 150)), a editor data object needs a **editor data class** for decoding. Every editor data class object should be added to the global **editor data class list**, returned by `get-the-editor-data-class-list`.

To store and load information about a snip or region in a editor:

1. derive new classes from `editor-data%` and `editor-data-class%`.

2. derive a new class from the `text%` or `pasteboard%` class, and override the `get-snip-data` and `set-snip-data` methods and/or the `get-region-data` and `set-region-data` methods.

When deriving the new `editor-data-class%` class, pick a new unique name to identify the encoded data. All names beginning with "wx" are reserved for internal use. By tagging extra data with a unique name, the normal editor content can be safely decoded in a editor that does not support the extra data.

8.2.2   Global Data: Headers and Footers

The editor file format provides for adding extra global data in special header and footer sections. To save and load special header and/or footer records:

1. Pick a name for each header/footer record. This name should not conflict with any other header/footer record name in use, and no one else should use these names. All names beginning with "wx" are reserved for internal use. By tagging extra header and footer records with a unique name, the file can be safely loaded under a system that does not support the records.

2. Derive a new class from the `text%` or `pasteboard%` class, and override the `write-headers-to-file`, `write-footers-to-file`, `read-header-from-file` and/or `read-footer-from-file` methods.

When a editor is saved, the methods `write-headers-to-file` and `write-footers-to-file` are invoked; this is when the derived `text%` or `pasteboard%` object has a chance to save records. To write a header/footer record, first invoke the `begin-write-header-footer-to-file` method, at which point the record name is provided. Once the record is written, call `end-write-header-footer-to-file`.

When a editor is loaded and a header/footer record is encountered, the `read-header-from-file` or `read-footer-from-file` method is invoked, with the record name as the argument. If the name matches a known record type, then the data can be loaded.

See also `write-headers-to-file` and `write-headers-to-file`.

8.3   End of Line Ambiguity

Because the editor can force a line break even when there is no carriage return item, a position alone does not always specify a graphic location for the caret. Consider the last position of a line which is soft-broken (i.e., no carriage return is present): there is no item between the last item of the line and the first item of the next line, so two graphic locations (one end-of-line and one start-of-line) map to the same position.

For this reason, position-setting and position-getting methods often have an extra argument. In the case of a position-setting method, the argument specifies whether the caret should be draw at the left or right side of

the page (in the event that the location is doubly defined); `#t` means that the caret should be drawn on the right side. Similarly, methods which calculate a position from a location will take an extra boxed boolean; the box is filled with `#t` if the position is ambiguous and it came from a right-side location, or `#f` otherwise.

8.4 Flattened Text

In plain text editors, there is a simple correlation between editor positions and characters. In a `editor<%>` object, this is not true much of the time, but it is still sometimes useful to just "get the text" of an editor.

There are two kinds of text available:

1. **Simple text**, where there is one character per item. Items which are characters are mapped to themselves, and all other items are mapped to a period. Line breaks are represented by carriage return characters (ASCII 13).

2. **Flattened text**, where each item can map to an arbitrary string. Items which are characters are still mapped to themselves, but more complicated items can be represented with a useful string, which is determined by the item's snip. Newlines are mapped to platform-specific character sequences (linefeed under X, carriage return under MacOS, and linefeed-carriage return under Windows). This is called "flattened" because the editor's items have been reduced to a linear sequence of characters.

8.5 Caret Ownership

Within a frame, only one object can contain the keyboard focus. This property must be maintained when a frame contains multiple editors in multiple displays, and when a single editor contains other editors as items.

When a editor has the keyboard focus, it will usually display the current selection, or a line indicating the insertion point; this line is called the **caret**.

When a editor contains other editors, it keeps track of caret ownership among the sub-editors it contains. When the caret is taken away from the main editor, it will take away caret ownership from the appropriate sub-editor.

When a editor or snip is drawn, an argument to the drawing method specifies whether the caret should be drawn with the data. This argument can be any of (in increasing order):

- `'no-caret` — The caret should not be drawn at all.

- `'show-inactive-caret` — The caret should be drawn as inactive; items may be identified as the local current selection, but the keyboard focus is elsewhere.

- `'show-caret` — The caret should be drawn to show keyboard focus ownership.

The `'show-inactive-caret` display mode is useful for showing selection ranges in text editors that do not have the focus. This `'show-inactive-caret` mode is distinct from `'no-caret` mode; when editors are embedded, only the locally-active editor shows its selection.

8.6 Cut and Paste Time Stamps

Methods of `editor<%>` that use the clipboard — including `copy`, `cut`, `paste`, and `do-edit-operation` — consume a time stamp argument. This time stamp is generally extracted from the `mouse-event%` or

`key-event%` object that triggered the clipboard action. X uses the time stamp to synchronize clipboard operations among the clipboard clients.

All instances of `event%` include a time stamp, which can be obtained using `get-time-stamp`.

If the time stamp is 0, it defauls to the current time. Using 0 as the time stamp almost always works fine, but it is considered bad manners under X.

8.7   Clickbacks

Clickbacks in a `text%` editor facilitate the creation of simple interactive objects, such as hyper-text. A clickback is defined by associating a callback function with a range of items in the editor. When a user clicks on the items in that range, the callback function is invoked. For example, a hyper-text clickback would associate a range with a callback function that changes the selection range in the editor.

By default, the callback function is invoked when the user releases the mouse button. The `set-clickback` method accepts an optional argument that causes the callback function to be invoked on the button press, instead. This behavior is useful, for example, for a clickback that creates a popup menu.

Note that there is no attempt to save clickback information when a file is saved, since a clickback will have an arbitrary procedure associated with it.

8.8   Internal Editor Locks

Instances of `editor<%>` have three levels of internal locking:

- write locking — When an editor is interally locked for writing, the abstract content of the editor cannot be changed. However, snips in a text editor can still be split and merged, and the text editor can be changed in ways that affects the flow of lines.

- flow locking — When a text editor is interally locked for reflowing, it is locked for writing and the actual snip content of the editor cannot change. Thus, no change can be made that would affect the flow of lines in the editor.

- read locking — When an editor is interally locked for reading, no operations can be performed on the editor. This extreme state is used only during callbacks to its snips while the editor is in a sensitive state.

The internal lock for an editor is *not* affected by calls to `lock`.

9. Editor Class Reference

9.1 Class Listing

Editors

```
editor<%>
 |- text%
 |- pasteboard%
```

Displays

```
editor-canvas%
editor-snip%
```

Snips

```
snip%
 |- string-snip%
 |   |- tab-snip%
 |- image-snip%
 |- editor-snip%
```

Administrators

```
editor-admin%
 |- editor-snip-editor-admin<%>
snip-admin%
```

Styles

```
add-color<%>
mult-color<%>
style<%>
style-delta%
style-list%
```

File Reading/Writing and Cut-and-Patse

```
editor-data%
editor-data-class%
editor-data-class-list<%>
editor-stream-in%
editor-stream-in-base%
```

```
editor-stream-in-string-base%
editor-stream-out%
editor-stream-out-base%
editor-stream-out-string-base%
snip-class%
snip-class-list<%>
```

Miscellaneous

```
editor-wordbreak-map%
keymap%
```

9.2   Buffer Method Table

The following is a table of methods in the `wx:media-buffer%`, `wx:media-edit%` and `wx:media-pasteboard%` arranged by category.

SELECTION & POSITIONS

```
add-selected in wx:media-pasteboard%
find-snip in wx:media-edit%
find-first-snip in wx:media-pasteboard%
find-next-selected-snip in wx:media-pasteboard%
flash-off in wx:media-edit%
flash-on in wx:media-edit%
get-anchor in wx:media-edit%
get-between-threshold in wx:media-edit%
get-end-position in wx:media-edit%
get-position in wx:media-edit%
get-snip-position in wx:media-edit%
get-snip-position-and-location in wx:media-edit%
get-start-position in wx:media-edit%
get-visible-position-range in wx:media-edit%
is-selected? in wx:media-pasteboard%
last-position in wx:media-edit%
line-end-position in wx:media-edit%
line-start-position in wx:media-edit%
move-position in wx:media-edit%
no-selected in wx:media-pasteboard%
position-line in wx:media-edit%
position-location in wx:media-edit%
scroll-to-position in wx:media-edit%
set-anchor in wx:media-edit%
set-between-threshold in wx:media-edit%
set-position in wx:media-edit%
set-position-bias-scroll in wx:media-edit%
set-selected in wx:media-pasteboard%
```

LOCATIONS & LINES

```
find-line in wx:media-edit%
find-position in wx:media-edit%
find-position-in-line in wx:media-edit%
find-snip in wx:media-pasteboard%
get-snip-location in wx:media-buffer%
get-snip-position-and-location in wx:media-edit%
get-visible-line-range in wx:media-edit%
global-to-local in wx:media-buffer%
last-line in wx:media-edit%
line-end-position in wx:media-edit%
line-length in wx:media-edit%
line-location in wx:media-edit%
line-start-position in wx:media-edit%
local-to-global in wx:media-buffer%
lower in wx:media-pasteboard%
move in wx:media-pasteboard%
move-to in wx:media-pasteboard%
position-line in wx:media-edit%
position-location in wx:media-edit%
raise in wx:media-pasteboard%
resize in wx:media-pasteboard%
set-after in wx:media-pasteboard%
set-before in wx:media-pasteboard%
set-position-bias-scroll in wx:media-edit%
```

PARAGRAPHS

```
last-paragraph in wx:media-edit%
line-paragraph in wx:media-edit%
paragraph-end-line in wx:media-edit%
paragraph-end-position in wx:media-edit%
paragraph-start-line in wx:media-edit%
paragraph-start-position in wx:media-edit%
position-paragraph in wx:media-edit%
```

INSERTING & DELETING

```
begin-edit-sequence in wx:media-buffer%
clear in wx:media-buffer%
delete in wx:media-edit%
delete in wx:media-pasteboard%
```

```
erase in wx:media-edit%
erase in wx:media-pasteboard%
end-edit-sequence in wx:media-buffer%
insert in wx:media-buffer%
insert in wx:media-edit%
insert in wx:media-pasteboard%
insert-box in wx:media-buffer%
insert-image in wx:media-buffer%
remove in wx:media-pasteboard%
```

CUT & PASTE & UNDO

```
clear-undos in wx:media-buffer%
copy in wx:media-buffer%
copy in wx:media-edit%
cut in wx:media-buffer%
cut in wx:media-edit%
get-max-undo-history in wx:media-buffer%
kill in wx:media-buffer%
paste in wx:media-buffer%
paste in wx:media-edit%
redo in wx:media-buffer%
set-max-undo-history in wx:media-buffer%
undo in wx:media-buffer%
```

STYLES

```
change-style in wx:media-buffer%
change-style in wx:media-edit%
change-style in wx:media-pasteboard%
get-style-list in wx:media-buffer%
get-tabs in wx:media-edit%
set-style-list in wx:media-buffer%
set-tabs in wx:media-edit%
```

TEXT

```
find-string in wx:media-edit%
find-string-all in wx:media-edit%
find-wordbreak in wx:media-edit%
get-character in wx:media-edit%
get-flattened-text in wx:media-buffer%
get-text in wx:media-edit%
get-wordbreak-map in wx:media-edit%
set-wordbreak-func in wx:media-edit%
set-wordbreak-map in wx:media-edit%
```

EVENTS & KEY MAPPING

```
add-buffer-functions in wx:media-buffer%
add-editor-functions in wx:media-edit%
add-pasteboard-functions in wx:media-pasteboard%
after-change-style in wx:media-edit%
after-delete in wx:media-edit%
after-delete in wx:media-pasteboard%
after-edit-sequence in wx:media-buffer%
after-insert in wx:media-edit%
after-insert in wx:media-pasteboard%
after-move-to in wx:media-pasteboard%
after-resize in wx:media-pasteboard%
after-set-position in wx:media-edit%
after-set-size-constraint in wx:media-edit%
get-focus-snip in wx:media-buffer%
get-keymap in wx:media-buffer%
on-char in wx:media-buffer%
on-change in wx:media-buffer%
on-change-style in wx:media-edit%
on-default-char in wx:media-buffer%
on-default-event in wx:media-buffer%
on-delete in wx:media-edit%
on-delete in wx:media-pasteboard%
on-edit-sequence in wx:media-buffer%
```

```
on-event in wx:media-buffer%
on-insert in wx:media-edit%
on-insert in wx:media-pasteboard%
on-local-char in wx:media-buffer%
on-local-event in wx:media-buffer%
on-move-to in wx:media-pasteboard%
on-resize in wx:media-pasteboard%
on-set-size-constraint in wx:media-edit%
set-keymap in wx:media-buffer%
```

VIEW & ADMINISTRATION

```
adjust-cursor in wx:media-buffer%
caret-hidden? in wx:media-edit%
get-admin in wx:media-buffer%
get-extent in wx:media-buffer%
get-dc in wx:media-buffer%
get-descent in wx:media-buffer%
get-dragable in wx:media-pasteboard%
get-inactive-focus-threshold in wx:media-buffer%
get-max-height in wx:media-buffer%
get-max-width in wx:media-buffer%
get-min-height in wx:media-buffer%
get-min-width in wx:media-buffer%
get-space in wx:media-buffer%
get-view-size in wx:media-buffer%
hide-caret in wx:media-edit%
invalidate-bitmap-cache in wx:media-buffer%
lock in wx:media-buffer%
needs-update in wx:media-buffer%
on-focus in wx:media-buffer%
on-paint in wx:media-buffer%
own-caret in wx:media-buffer%
refresh in wx:media-buffer%
resized in wx:media-buffer%
scroll-to in wx:media-buffer%
scroll-to-position in wx:media-edit%
set-admin in wx:media-buffer%
set-autowrap-bitmap in wx:media-edit%
set-caret-owner in wx:media-buffer%
set-cursor in wx:media-edit%
set-dragable in wx:media-pasteboard%
set-inactive-focus-threshold in wx:media-buffer%
set-max-height in wx:media-buffer%
set-max-width in wx:media-buffer%
set-min-height in wx:media-buffer%
set-min-width in wx:media-buffer%
size-cache-invalid in wx:media-buffer%
```

FILE LOADING & SAVING

```
after-load-file in wx:media-buffer%
after-save-file in wx:media-buffer%
begin-write-header-footer-to-file in wx:media-buffer%
end-write-header-footer-to-file in wx:media-buffer%
get-file in wx:media-buffer%
get-filename in wx:media-buffer%
get-file-format in wx:media-edit%
get-region-data in wx:media-edit%
get-snip-data in wx:media-buffer%
insert-file in wx:media-buffer%
load-file in wx:media-buffer%
modified? in wx:media-buffer%
on-load-file in wx:media-buffer%
on-save-file in wx:media-buffer%
put-file in wx:media-buffer%
read-footer-from-file in wx:media-buffer%
read-from-file in wx:media-buffer%
read-from-file in wx:media-edit%
read-header-from-file in wx:media-buffer%
save-file in wx:media-buffer%
set-filename in wx:media-buffer%
set-file-format in wx:media-edit%
set-modified in wx:media-buffer%
set-region-data in wx:media-edit%
set-snip-data in wx:media-buffer%
```

```
write-headers-to-file in wx:media-buffer%
write-footers-to-file in wx:media-buffer%
write-to-file in wx:media-buffer%
write-to-file in wx:media-edit%
```

MENUS

```
append-edit-items in wx:media-buffer%
append-font-items in wx:media-buffer%
do-edit in wx:media-buffer%
do-font in wx:media-buffer%
```

OTHER

```
copy-self in wx:media-buffer%
copy-self-to in wx:media-buffer%
copy-self-to in wx:media-edit%
copy-self-to in wx:media-pasteboard%
set-clickback in wx:media-edit%
on-new-image-snip in wx:media-buffer%
on-new-tab-snip in wx:media-edit%
on-new-text-snip in wx:media-edit%
```

9.3   `add-color<%>`

A `add-color<%>` object is used to additively change the RGB values of a `color%` object. A `add-color<%>` object only exists within a `style-delta%` object.

See also `get-foreground-add` and `get-background-add`.

get

Gets all of the additive values.

- (send *an-add-color* `get` *r g b*) ⇒ void
  r : boxed exact integer in [-1000, 1000]
  g : boxed exact integer in [-1000, 1000]
  b : boxed exact integer in [-1000, 1000]

  The *r* box is filled with the additive value for the red component of the color. The *g* box is filled with the additive value for the green component of the color. The *b* box is filled with the additive value for the blue component of the color.

get-b

Gets the additive value for the blue component of the color.

- (send *an-add-color* `get-b`) ⇒ exact integer in [-1000, 1000]

get-g

Gets the additive value for the green component of the color.

- (send *an-add-color* `get-g`) ⇒ exact integer in [-1000, 1000]

get-r

Gets the additive value for the red component of the color.

- (send *an-add-color* `get-r`) ⇒ exact integer in [-1000, 1000]

set

Sets all of the additive values.

- (send *an-add-color* `set` *r g b*) ⇒ void
  r : exact integer in [-1000, 1000]
  g : exact integer in [-1000, 1000]
  b : exact integer in [-1000, 1000]

set-b

Sets the additive value for the blue component of the color.

- (send *an-add-color* `set-b` *v*) ⇒ void
  v : exact integer in [-1000, 1000]

set-g

Sets the additive value for the green component of the color.

- (send *an-add-color* `set-g` *v*) ⇒ void
  v : exact integer in [-1000, 1000]

set-r

Sets the additive value for the red component of the color.

- (send *an-add-color* `set-r` *v*) ⇒ void
  v : exact integer in [-1000, 1000]

9.4   `editor<%>`

The `editor<%>` interface is implemented by `text%` and `pasteboard%`.

See **??**§ for a table of editor methods sorted by kind of functionality.

add-canvas

Adds a canvas to this editor's list of displaying canvases. (See `get-canvases`.)

Normally, this method is called only by `set-editor` in `editor-canvas%` .

- (send *an-editor* `add-canvas` *canvas*) ⇒ void
  canvas : `editor-canvas%` object

add-undo

Adds an undoer procedure to the editor's undo stack. If an undo is currently being performed, the undoer is added to the editor's redo stack. The undoer is called by the system when it is undoing (or redoing) changes to a editor, and when this undoer is the first item on the undo (or redo) stack.

The system automatically installs undo records to undo built-in editor operations, such as inserts, deletes, and font changes. Install an undoer only when it is necessary to maintain state or handle operations that are not built-in. For example, in a program where the user can assign labels to snips in a pasteboard, the program should install an undoer to revert a label change. Thus, when a user changes a snip's label and then selects Undo (from a standard menu bar), the snip's label will revert as expected. In contrast, there is no need to install an undoer when the user moves a snip by dragging it, because the system installs an appropriate undoer automatically.

After an undoer returns, the undoer is popped off the editor's undo (or redo) stack; if the return value is true, then the next undoer is also executed as part of the same undo (or redo) step. The undoer should return true if the action being undone was originally performed as part of a `begin-edit-sequence` and `end-edit-sequence` sequence. The return value should also be true if the undone action was implicitly part

of a sequence. To extend the previous example, if a label change is paired with a move to realign the snip, then the label-change undoer should be added to the editor *after* the call to `move`, and it should return `#t` when it is called. As a result, the move will be undone immediately after the label change is undone. (If the opposite order is needed, use `begin-edit-sequence` and `end-edit-sequence` to create an explicit sequence.)

The system adds undoers to an editor (in response to other method calls) without calling this method.

- (send *an-editor* `add-undo` *undoer*) ⇒ void
    undoer : procedure of zero arguments

`adjust-cursor`

Gets a cursor to be used in the editor's display. If the return value is `#f`, a default cursor is used.

See also `set-cursor`.

- (send *an-editor* `adjust-cursor` *event*) ⇒ cursor% object or #f
    event : mouse-event% object

If an overriding cursor has been installed with `set-cursor`, then the installed cursor is returned.

Otherwise, if the event is a dragging event, a snip in the editor has the focus, and the snip's `adjust-cursor` method returns a cursor, that cursor is returned.

Otherwise, if the cursor is over a snip and the snip's `adjust-cursor` method returns a cursor, that cursor is returned.

Otherwise, if a cursor has been installed with `set-cursor`, then the installed cursor is returned.

Otherwise, if the cursor is over a clickback region in an editor, an arrow cursor is returned.

Finally, if none of the above cases apply, a default cursor is returned. For a text editor, the default cursor is an I-beam. For a pasteboard editor, the default cursor is an arrow.

`after-edit-sequence`

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

See also `on-edit-sequence`.

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

`after-load-file`

Called just after the editor is loaded from a file. The argument to the method specifies whether the load was sucessful or not. See also `can-load-file?` and `on-load-file`.

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

`after-save-file`

Called just after the editor is saved to a file. The argument to the method specifies whether the save was sucessful or not. See also `can-save-file?`and `on-save-file`.

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

auto-wrap

Enables or disables automatically calling `set-max-width` in response to `on-display-size`, or gets the state of auto-wrapping. For text editors, this has the effect of wrapping the editor's contents to fit in a canvas displaying the editor (the widest one if multiple canvases display the editor). For pasteboard editors, "auto-wrapping" merely truncates the area of the pasteboard to match its canvas display.

Auto-wrapping is initially disabled.

- (send *an-editor* `auto-wrap`) ⇒ boolean

    Returns `#t` if auto-wrapping is enabled, `#f` otherwise.

- (send *an-editor* `auto-wrap` *auto-wrap?*) ⇒ void
    auto-wrap? : boolean

    Enables auto-wrapping if *auto-wrap?* is true, disables auto-wrapping otherwise. The `on-display-size` method is called immediately to update the editor's maximum width.

begin-edit-sequence

The `begin-edit-sequence` and `end-edit-sequence` methods are used to bracket a set of editor modifications so that the results are all displayed at once. The commands may be nested arbitrarily deep. Using these functions can greatly speed up displaying the changes.

When a editor contains other editors, using `begin-edit-sequence` and `end-edit-sequence` on the main editor brackets some changes to the sub-editors as well, but it is not as effective when a sub-editor changes as calling `begin-edit-sequence` and `end-edit-sequence` for the sub-editor.

See also `refresh-delayed?`.

- (send *an-editor* `begin-edit-sequence` *undoable?*) ⇒ void
    undoable? = `#t` : boolean

    If the *undoable?* flag is `#f`, then the changes made in the sequence cannot be reversed through the `undo` method. This flag is only effective for the outermost `begin-edit-sequence` when nested sequences are used.

begin-write-header-footer-to-file

This method must be called before writing any special header data to a stream.

See section 8.2 (page 150) and `write-headers-to-file` for more information.

- (send *an-editor* `begin-write-header-footer-to-file` *f* *name* *buffer*) ⇒ void
    f : `editor-stream-out%` object
    name : string
    buffer : boxed exact integer

    The *name* string must be a unique name that can be used by a header reader to recognize the data. This method will store a value in *buffer* that should be passed on to `end-write-header-footer-to-file`.

`blink-caret`

Tells the editor to blink the selection caret. This method is called periodically when the editor's display has the keyboard focus.

- (send *an-editor* `blink-caret`) ⇒ void

    Propagates the flag to any snip with the editor-local focus.

`can-do-edit-operation?`

Checks whether a generic edit command would succeed for the editor. This check is especially useful for enabling and disabling menus on demand.

- (send *an-editor* `can-do-edit-operation?` *op recursive?*) ⇒ bool
    op : symbol   in   '(undo redo clear cut copy paste kill select-all
          insert-text-box insert-pasteboard-box insert-image)
    recursive? = #t : boolean

    See `do-edit-operation` for information about the *op* and *recursive?* arguments.

`can-load-file?`

Called just before the editor is loaded from a file. If the return value is #f, the file is not loaded. See also `on-load-file` and `after-load-file`.

- (send *an-editor* `can-load-file?` *filename format*) ⇒ boolean
    filename : string
    format : symbol in '(guess standard text text-force-cr same copy)

    The *filename* argument is the name the file will be loaded from. See `load-file` for information about *format*.

`can-save-file?`

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

- (send *an-editor* `can-save-file?` *filename format*) ⇒ boolean
    filename : string
    format : symbol in '(guess standard text text-force-cr same copy)

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

`change-style`

Changes the style for items in the editor.

The style within an editor can be changed by the system (in response to other method calls), and such changes do not go through this method; use `on-change-style` in `text%` to monitor style changes.

- (send *an-editor* `change-style` *delta*) ⇒ void
    delta : style-delta% object

Changes the style of the selected items by applying a style delta.

To change a large collection of snips from one style to another style, consider providing a `style<%>` instance rather than a `style-delta%` instance. Otherwise, `change-style` must convert the `style-delta%` instance to the `style<%>` instance for every snip; this conversion consumes both time and (temporary) memory.

- (send *an-editor* `change-style` *style*) ⇒ void
    style : `style<%>` object

Changes the style of the selected items to a specific style. The editor's style list must contain *style*, otherwise the style is not changed. See also `convert`.

clear

Deletes the currently selected items.

The content of an editor can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-delete` in `text%` or `on-delete` in `pasteboard%` to monitor content deletions changes.

- (send *an-editor* `clear`) ⇒ void

clear-undos

Destroys the undo history of the editor.

- (send *an-editor* `clear-undos`) ⇒ void

copy

Copies items into the clipboard.

The system may execute a copy (in response to other method calls) without calling this method. To extend or re-implement copying, override the `do-copy` in `text%` or `do-copy` in `pasteboard%` method of an editor.

- (send *an-editor* `copy` *extend?*   *time*) ⇒ void
    extend? = `#f` : boolean
    time = 0 : exact integer

Copies the selected items into the clipboard. If *extend?* is not `#f`, the old clipboard contents are appended.

See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

copy-self

Creates a new editor with the same properties as this one.

- (send *an-editor* `copy-self`) ⇒ `text%` or `pasteboard%` object

After an editor is created (either a `text%` or `pasteboard%` instance, as appropriate), the new editor is passed to `copy-self-to`.

`copy-self-to`

Copies the properties of this editor into an existing editor.

- (send *an-editor* `copy-self-to` *dest*) ⇒ void
    dest : `text%` or `pasteboard%` object

  Each snip in this editor is copied and inserted into *dest*. In addition, this editor's filename, maximum undo history setting, keymap, interactive caret threshold, and overwrte-styles-on-load settings are installed into *dest*. This editor's style list is copied and the copy is installed as the style list for *dest*.

`cut`

Copies and then deletes items in the editor.

The system may execute a cut (in response to other method calls) without calling this method. To extend or re-implement the copying portion of the cut, override the `do-copy` in `text%` or `do-copy` in `pasteboard%` method of an editor. To monitor deletions in an editor, override `on-delete` in `text%` or `on-delete` in `pasteboard%`.

- (send *an-editor* `cut` *extend?*   *time*) ⇒ void
    extend? = `#f` : boolean
    time = `0` : exact integer

  Copies and then deletes the currently selected items. If *extend?* is not `#f`, the old clipboard contents are appended.

  See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`dc-location-to-editor-location`

Converts the given coordinates from top-level display coordinates (usually canvas coordinates) to editor location coordinates. The same calculation is performed by `global-to-local`.

The result is only valid when the editor is displayed (see section 8.1 (page 148)).

See also `editor-location-to-dc-location`.

- (send *an-editor* `dc-location-to-editor-location` *x y*) ⇒ two real numbers
    x : real number
    y : real number

  Returns the equivalent of *x* and *y* translated from DC coordinates to editor drawing coordinates.

`do-edit-operation`

Performs a generic edit command.

- (send *an-editor* `do-edit-operation` *op recursive?*   *time*) ⇒ void
    op : symbol   in   `'(undo redo clear cut copy paste kill select-all insert-text-box insert-pasteboard-box insert-image)`
    recursive? = `#t` : boolean
    time = `0` : exact integer

  The *op* argument must be a valid edit command, one of:

- – 'undo — undoes the last operation
- – 'redo — undoes the last undo
- – 'clear — deletes the current selection
- – 'cut — cuts
- – 'copy — copies
- – 'paste — pastes
- – 'kill — cuts to the end of the current line, or cuts a newline if there is only whitespace between the selection and end of line
- – 'select-all — selects everything in the editor
- – 'insert-text-box — inserts a text editor as an item in this editor; see also `on-new-box` .
- – 'insert-pasteboard-box — inserts a pasteboard editor as an item in this editor; see also `on-new-box` .
- – 'insert-image — gets a filename from the user and inserts the image as an item in this editor; see also `on-new-image-snip` .

If *recursive?* is not `#f`, then the command is passed on to any active snips of this editor (i.e., snips which own the caret).

See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`editor-location-to-dc-location`

Converts the given coordinates from editor location coordinates to top-level display coordinates (usually canvas coordinates). The same calculation is performed by `local-to-global`.

The result is only valid when the editor is displayed (see section 8.1 (page 148)).

See also `dc-location-to-editor-location`.

- (send *an-editor* `editor-location-to-dc-location` *x y*) $\Rightarrow$ two real numbers
    x : real number
    y : real number

Returns the equivalent of *x* and *y* translated from editor coordinates to DC drawing coordinates.

`end-edit-sequence`

See `begin-edit-sequence`.

- (send *an-editor* `end-edit-sequence`) $\Rightarrow$ void

`end-write-header-footer-to-file`

This method must be called after writing any special header data to a stream.

- (send *an-editor* `end-write-header-footer-to-file` *f buffer-value*) $\Rightarrow$ void
    f : editor-stream-out% object
    buffer-value : exact integer

The *buffer-value* argument must be the value put in the *buffer* argument box by `begin-write-header-footer-to-file`.

See section 8.2 (page 150) and `write-headers-to-file` for more information.

`find-first-snip`

Returns the first snip in the editor, or `#f` if the editor is empty. To get all of the snips in the editor, use the `next` in `snip%` on the resulting snip.

The first snip in a text editor is the one at position 0. The first snip in a pasteboard is the frontmost snip. (See section 8.1 (page 148) for information about snip order in pasteboards.)

- (send *an-editor* `find-first-snip`) ⇒ `snip%` object or `#f`

`find-scroll-line`

Maps a vertical location within the editor to a vertical scroll position.

For `text%` objects: Calling this method may force the recalculation of location information, even if the editor currently has delayed refreshing (see `refresh-delayed?`). The result is only valid when the editor is displayed (see section 8.1 (page 148)).

- (send *an-editor* `find-scroll-line` *location*) ⇒ exact non-negative integer
    location : real number

`get-active-canvas`

If the editor is displayed in a canvas, this method returns the canvas that most recently had the keyboard focus (while the editor was displayed). If no such canvas exists, `#f` is returned.

- (send *an-editor* `get-active-canvas`) ⇒ `editor-canvas%` object or `#f`

`get-admin`

Returns the `editor-admin%` object currently managing this editor or `#f` if the editor is not displayed.

- (send *an-editor* `get-admin`) ⇒ `editor-admin%` object or `#f`

`get-canvas`

If `get-active-canvas` returns a canvas, that canvas is also returned by this method. Otherwise, if `get-canvases` returns a non-empty list, the first canvas in the list is returned, otherwise `#f` is returned.

- (send *an-editor* `get-canvas`) ⇒ `editor-canvas%` object or `#f`

`get-canvases`

Returns a list of canvases displaying the editor. An editor may be displayed in multiple canvases and no other kind of display, or one instance of another kind of display and no canvases. If the editor is not displayed or the editor's current display is not a canvas, `null` is returned.

- (send *an-editor* `get-canvases`) ⇒ list of `editor-canvas%` objects

`get-dc`

Typically used (indirectly) by snip objects belonging to the editor. Returns a destination drawing context which is suitable for determining display sizing information, or #f if the editor is not displayed.

- (send *an-editor* get-dc) ⇒ dc<%> object or #f

`get-descent`

Returns the font descent for the editor. This method is primarily used when an editor is an item within another editor.

The result is only valid when the editor is displayed (see section 8.1 (page 148)). For text% objects, calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see refresh-delayed?).

- (send *an-editor* get-descent) ⇒ non-negative real number

`get-extent`

Gets the current extent of the editor's graphical representation.

The result is only valid when the editor is displayed (see section 8.1 (page 148)). For text% objects, calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see refresh-delayed?).

- (send *an-editor* get-extent *w h*) ⇒ void
    w : boxed non-negative real number or #f
    h : boxed non-negative real number or #f

  The *w* box is filled with the editor's width, unless *w* is #f. The *h* box is filled with the editor's height, unless *h* is #f.

`get-file`

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

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

  Calls the global get-fileprocedure.

  If the editor is displayed in a single canvas, then the canvas's top-level frame is used as the parent for the file dialog. Otherwise, the file dialog will have no parent.

`get-filename`

Returns the path name of the last file saved from or loaded into this editor, #f if the editor has no filename.

- (send *an-editor* get-filename *temp*) ⇒ string or #f
    temp = #f : boxed boolean or #f

  The *temp* box is filled with #t if the filename is temporary or #f otherwise.

`get-flattened-text`

Returns the contents of the editor in text form. See section 8.4 (page 152) for a discussion of flattened vs. non-flattened text.

- (send *an-editor* `get-flattened-text`) ⇒ string

`get-focus-snip`

Returns the snip within the editor that gets the keyboard focus when the editor has the focus, or `#f` if the editor does not delegate the focus.

The returned snip might be an `editor-snip%` object. In that case, the embedded editor might delegate the focus to one of its own snips. However, the `get-focus-snip` method returns only the `editor-snip%` object, because it is the focus-owning snip within the immediate editor.

See also `set-caret-owner` .

- (send *an-editor* `get-focus-snip`) ⇒ `snip%` object or `#f`

`get-inactive-caret-threshold`

Returns the threshold for painting an inactive selection. This threshold is compared with the *draw-caret* argument to `refresh` and if the argument is as least as large as the threshold (but larger than `'show-caret`), the selection is drawn as inactive.

See also `set-inactive-caret-threshold` and section 8.5 (page 152).

- (send *an-editor* `get-inactive-caret-threshold`) ⇒ symbol in `'(no-caret show-inactive-caret show-caret)`

`get-keymap`

Returns the main keymap currently used by the editor.

- (send *an-editor* `get-keymap`) ⇒ `keymap%` object or `#f`

`get-load-overwrites-styles`

Reports whether named styles in the current style list are replaced by `load-file` when the loaded file contains style specifications.

See also `set-load-overwrites-styles`.

- (send *an-editor* `get-load-overwrites-styles`) ⇒ boolean

`get-max-height`

Gets the maximum display height for the contents of the editor; zero or `'none` indicates that there is no maximum.

- (send *an-editor* get-max-height) ⇒ non-negative real number or 'none

get-max-undo-history

Returns the maximum number of undoables that will be remembered by the editor. Note that undoables are counted by insertion, deletion, etc. events, not by the number of times that undo can be called; a single undo call often reverses multiple events at a time (such as when the user types a stream of characters at once).

- (send *an-editor* get-max-undo-history) ⇒ exact integer in [0, 100000]

get-max-view-size

Returns the maximum visible area into which the editor is currently being displayed, according to the editor's administrators. If the editor has only one display, the result is the same as for get-view-size. Otherwise, the maximum width and height of all the editor's displaying canvases is returned.

The result is only valid when the editor is displayed (see section 8.1 (page 148)).

If the display is an editor canvas, see also reflow-container.

- (send *an-editor* get-max-view-size) ⇒ two non-negative real numbers

get-max-width

Gets the maximum display width for the contents of the editor; zero or 'none indicates that there is no maximum. In a text editor, zero of 'none disables automatic line breaking.

- (send *an-editor* get-max-width) ⇒ non-negative real number or 'none

get-min-height

Gets the minimum display height for the contents of the editor; zero or 'none indicates that there is no minimum.

- (send *an-editor* get-min-height) ⇒ non-negative real number or 'none

get-min-width

Gets the minimum display width for the contents of the editor; zero or 'none indicates that there is no minimum.

- (send *an-editor* get-min-width) ⇒ non-negative real number or 'none

get-paste-text-only

If the result is #t, then the editor accepts only plain-text data from the clipboard. If the result is #f, the editor accepts both text and snip data from the clipboard.

- (send *an-editor* get-paste-text-only) ⇒ boolean

get-snip-data

Gets extra data associated with a snip (e.g., location information in a pasteboard) or returns `#f` is there is no information. See section 8.2.1 (page 151) for more information.

- (send *an-editor* `get-snip-data` *thesnip*) $\Rightarrow$ `editor-data%` object or `#f`
    thesnip : `snip%` object

    Returns `#f`.

get-snip-location

Gets the graphical location of the given snip. If the snip is found in the editor, `#t` is returned; otherwise, `#f` is returned.

The result is only valid when the editor is displayed (see section 8.1 (page 148)). For `text%` objects, calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (send *an-editor* `get-snip-location` *thesnip* *x* *y* *bottom-right?*) $\Rightarrow$ boolean
    thesnip : `snip%` object
    x = `#f` : boxed real number or `#f`
    y = `#f` : boxed real number or `#f`
    bottom-right? = `#f` : boolean

    The x box is filled with the x-coordinate of the snip's location, unless x is `#f`. The y box is filled with the y-coordinate of the snip's location, unless y is `#f`.

    If *bottom-right?* is not `#f`, the values in the x and y boxes are for the snip's bottom right corner instead of its top-left corner.

get-space

Returns the maximum font space for the editor. This method is primarily used when an editor is an item within another editor.

The result is only valid when the editor is displayed (see section 8.1 (page 148)). For `text%` objects, calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (send *an-editor* `get-space`) $\Rightarrow$ non-negative real number

get-style-list

Returns the style list currently in use by the editor.

- (send *an-editor* `get-style-list`) $\Rightarrow$ `style-list%` object

get-view-size

Returns the visible area into which the editor is currently being displayed (according to the editor's administrator). See also `get-view` .

The result is only valid when the editor is displayed (see section 8.1 (page 148)).

If the display is an editor canvas, see also `reflow-container`.

- (send *an-editor* `get-view-size` *w h*) ⇒ void
    w : boxed non-negative real number or `#f`
    h : boxed non-negative real number or `#f`

  The *w* box is filled with the visible area width, unless *w* is `#f`. The *h* box is filled with the visible area height, unless *h* is `#f`.

`global-to-local`

Converts the given coordinates from top-level display coordinates (usually canvas coordinates) to editor location coordinates. The same calculation is performed by `dc-location-to-editor-location`.

The result is only valid when the editor is displayed (see section 8.1 (page 148)).

See also `local-to-global`.

- (send *an-editor* `global-to-local` *x y*) ⇒ void
    x : boxed real number or `#f`
    y : boxed real number or `#f`

  The *x* box is filled with the translated x-coordiante of the value initially in *x*, unless *x* is `#f`. The *y* box is filled with the translated x-coordiante of the value initially in *y*, unless *y* is `#f`.

`insert`

Inserts data into the editor.

The content of an editor can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-insert` in `text%` or `on-insert` in `pasteboard%` to monitor content additions changes.

- (send *an-editor* `insert` *snip*) ⇒ void
    snip : `snip%` object

  Inserts a snip into the editor. A snip cannot be inserted into multiple editors or multiple times within a single editor.

`insert-box`

Inserts a box (a sub-editor) into the editor.

The content of an editor can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-insert` in `text%` or `on-insert` in `pasteboard%` to monitor content additions changes.

- (send *an-editor* `insert-box` *type*) ⇒ void
    type = `'text` : symbol in `'(text pasteboard)`

  Calls `on-new-box`, passing along *type* and inserts the resulting snip into the editor.

171

insert-file

Inserts a file into the editor (at the current selection position in `text%` editors).

The content of an editor can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-insert` in `text%` or `on-insert` in `pasteboard%` to monitor content additions changes.

- (send *an-editor* `insert-file` *filename format show-errors?*) ⇒ boolean
    filename : string
    format = `'guess` : symbol in `'(guess standard text text-force-cr same copy)`
    show-errors? = `#t` : boolean

  For more information on file formats, see `load-file`. If *show-errors?* is `#f`, error messages in loading the file (printed to stdout) are suppressed.

insert-image

Inserts an image into the editor.

The content of an editor can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-insert` in `text%` or `on-insert` in `pasteboard%` to monitor content additions changes.

- (send *an-editor* `insert-image` *filename type relative-path? inline?*) ⇒ void
    filename = `#f` : string or `#f`
    type = `'unknown` : symbol in `'(unknown gif jpeg xbm xpm bmp pict)`
    relative-path? = `#f` : boolean
    inline? = `#t` : boolean

  If *filename* is `#f`, then the user is queried for a filename. The *kind* must one of the symbols that can be passed to `load-file`.

  After the filename has been determined, an image is created by calling `on-new-image-snip`. See also `image-snip%`.

invalidate-bitmap-cache

When `on-paint` is overridden, call this method when the state of `on-paint`'s drawing changes.

- (send *an-editor* `invalidate-bitmap-cache` *x y width height*) ⇒ void
    x = `0.0` : real number
    y = `0.0` : real number
    width = `'end` : non-negative real number or `'end`
    height = `'end` : non-negative real number or `'end`

  The *x*, *y*, *width*, and *height* arguments specify the area that needs repainting in editor coordinates. If *width*/*height* is `'end`, then the total height/width of the editor (as reported by `get-extent`) is used. Note that the editor's size can be smaller than the visible region of its display.

is-locked?

Returns `#t` if the editor is currently locked, `#f` otherwise. See `lock` for more information.

- (send *an-editor* `is-locked?`) ⇒ boolean

is-modified?

Returns #t is the editor has been modified since the last save or load, #f otherwise.

- (send *an-editor* is-modified?) ⇒ boolean

kill

In a text editor, cuts to the end of the current line, or cuts a newline if there is only whitespace between the selection and end of line. Multiple consective kills are appended. In a pasteboard editor, cuts the current selection.

See also cut.

The content of an editor can be changed by the system in response to other method calls, and such changes do not go through this method; use on-delete in text% or on-delete in pasteboard% to monitor content deletions changes.

- (send *an-editor* kill *time*) ⇒ void
    time = 0 : exact integer

    See section *8.6* (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an exn:application:mismatch exception is raised.

load-file

Loads a file into the editor, return #t if successful, #f otherwise.

The filename used to load the file can be retrieved with get-filename. For a text% instance, the format can be retrieved with get-file-format.

See also on-load-file, after-load-file, can-load-file?, and set-load-overwrites-styles.

- (send *an-editor* load-file *filename format show-errors?*) ⇒ boolean
    filename = #f : string or #f
    format = 'guess : symbol in '(guess standard text text-force-cr same copy)
    show-errors? = #t : boolean

    If *filename* is #f, then the internally stored filename will be used; if *filename* is "" or if the internal name is unset or temporary, then the user will be prompted for a name.

    The possible values for *format* are listed below. A single set of *format* values are used for loading and saving files:
        – 'guess — guess the format based on extension and/or contents; when saving a file, this is the same as 'standard
        – 'standard — read/write a standard file
        – 'text — read/write a text file (text% only)
        – 'text-force-cr — read/write a text file (text% only); when writing, change automatic newlines (from word-wrapping) into real carriage returns
        – 'same — read in whatever format was last loaded or saved
        – 'copy — write using whatever format was last loaded or saved, but do not change the modification flag or remember *filename* (saving only)

    In a text% instance, the format returned from get-file-format is always one of 'standard, 'text, or 'text-force-cr.

If *show-errors?* is `#f`, then error messages reporting load errors (printed to stdout) are suppressed.

`local-to-global`

Converts the given coordinates from editor location coordinates to top-level display coordinates (usually canvas coordinates). The same calculation is performed by `editor-location-to-dc-location`.

The result is only valid when the editor is displayed (see section 8.1 (page 148)).

See also `global-to-local`.

- (send *an-editor* `local-to-global` *x y*) $\Rightarrow$ void
  x : boxed real number
  y : boxed real number

  The *x* box is filled with the translated x-coordiante of the value initially in *x*, unless *x* is `#f`. The *y* box is filled with the translated x-coordiante of the value initially in *y*, unless *y* is `#f`.

`lock`

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

See also `is-locked?`.

This method does not affect internal locks, as discussed in section 8.8 (page 153).

- (send *an-editor* `lock` *lock?*) $\Rightarrow$ void
  lock? : boolean

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

`needs-update`

Typically called (indirectly) by a snip within the editor to force the editor to be redrawn.

For `text%` objects, calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (send *an-editor* `needs-update` *snip localx localy w h*) $\Rightarrow$ void
  snip : `snip%` object
  localx : real number
  localy : real number
  w : non-negative real number
  h : non-negative real number

  The *localx*, *localy*, *width*, and *height* arguments specify the area that needs repainting in the coordinate system of *snip*.

`num-scroll-lines`

Reports the number of scroll positions available within the editor.

For `text%` objects: Calling this method may force the recalculation of location information, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

- (`send` *an-editor* `num-scroll-lines`) $\Rightarrow$ exact non-negative integer

on-change

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

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

- (`send` *an-editor* `on-change`) $\Rightarrow$ void

on-char

Handles keyboard input to the editor.

Consider overriding `on-local-char` or `on-default-char` instead of this method.

- (`send` *an-editor* `on-char` *event*) $\Rightarrow$ void
  event : `key-event%` object

  Either passes this event on to a caret-owning snip or calls `on-local-char` .

on-default-char

Called by `on-local-char` when the event is *not* handled by a caret-owning snip or by the keymap.

- (`send` *an-editor* `on-default-char` *event*) $\Rightarrow$ void
  event : `key-event%` object

  Does nothing.

on-default-event

Called by `on-local-event` when the event is *not* handled by a caret-owning snip or by the keymap.

- (`send` *an-editor* `on-default-event` *event*) $\Rightarrow$ void
  event : `mouse-event%` object

  Does nothing. See also `on-default-event` in `pasteboard%`.

on-display-size

This method is called by the editor's display whenever the display's size (as reported by `get-view-size`) changes.

- (`send` *an-editor* `on-display-size`) $\Rightarrow$ void

If automatic wrapping is enabled (see `auto-wrap`) then `set-max-width` is called with the maximum width of all of the editor's canvases (according to the administrators; `call-as-primary-owner` in `editor-canvas%` is used with each canvas to set the administrator and get the view size). If the editor is displayed but not in a canvas, the unique width is obtained from the editor's administrator (there is only one). If the editor is not displayed, the editor's maximum width is not changed.

on-edit-sequence

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

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

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

- (send *an-editor* `on-edit-sequence`) ⇒ void

on-event

Handles mouse input to the editor. The event's x and y coordinates are in the display's coordinate system; use the administrator's `get-dc` method to obtain translation arguments (or use `dc-location-to-editor-location`).

Consider overriding `on-local-event` or `on-default-event` instead of this method.

- (send *an-editor* `on-event` *event*) ⇒ void
    event : `mouse-event%` object

    Either passes this event on to a caret-owning snip or calls `on-local-event` .

on-focus

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

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

on-load-file

Called just before the editor is loaded from a file, after calling `can-load-file?` to verify that the load is allowed. See also `after-load-file`.

- (send *an-editor* `on-load-file` *filename format*) ⇒ void
    filename : string
    format : symbol in '(guess standard text text-force-cr same copy)

The *filename* argument is the name the file will be loaded from. See `load-file` for information about *format*.

on-local-char

Called by `on-char` when the event is *not* handled by a caret-owning snip.

Consider overriding `on-default-char` instead of this method.

- (`send` *an-editor* `on-local-char` *event*) ⇒ void
  event : `key-event%` object

Either lets the keymap handle the event or calls `on-default-char` .

on-local-event

Called by `on-event` when the event is *not* handled by a caret-owning snip.

Consider overriding `on-default-event` instead of this method.

- (`send` *an-editor* `on-local-event` *event*) ⇒ void
  event : `mouse-event%` object

Either lets the keymap handle the event or calls `on-default-event` .

on-new-box

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

- (`send` *an-editor* `on-new-box` *type*) ⇒ `snip%` object
  type : symbol in '(text pasteboard)

Creates a `editor-snip%` with either a sub-editor from `text%` or sub-pasteboard from `pasteboard%`, depending on whether *type* is 'text or 'pasteboard. The keymap (see `keymap%`) and style list (see `style-list%`) for of the new sub-editor are set to the keymap and style list of this editor.

on-new-image-snip

Creates and returns a new instance of `image-snip%` for `insert-image`.

- (`send` *an-editor* `on-new-image-snip` *filename kind relative-path? inline?*) ⇒ `image-snip%` object
  filename : string or `#f`
  kind : symbol in '(unknown gif jpeg xbm xpm bmp pict)
  relative-path? : boolean
  inline? : boolean

Returns (`make-object` `image-snip%` *filename kind reltaive-path? inline?*).

on-paint

Provides a way to add arbitrary graphics to an editors's display. This method is called just before and just after every painting of the editor.

The `on-paint` method, together with the snips' `draw` methods, must be able to draw the entire state of an editor. Never paint directly into an editor's display canvas except from within `on-paint` or `draw`. Instead, put all extra drawing code within `on-paint` and call `invalidate-bitmap-cache` when part of the display needs to be repainted.

The `on-paint` method must not make any assumptions about the state of the drawing context (e.g., the current pen), except that the clipping region is already set to something appropriate. Before `on-paint` returns, it must restore any drawing context settings that it changes.

The editor is internally locked for writing and reflowing during a call to this method (see also section 8.8 (page 153)).

- (send *an-editor* `on-paint` *before?   dc left top right bottom dx dy draw-caret*) ⇒ void
    before? : boolean
    dc : `dc<%>` object
    left : real number
    top : real number
    right : real number
    bottom : real number
    dx : real number
    dy : real number
    draw-caret : symbol in '(`no-caret show-inactive-caret show-caret`)

  The *before?* argument is `#t` when the method is called just before a painting the contents of the editor or `#f` when it is called after painting. The *left*, *top*, *right*, and *bottom* arguments specify which region of the editor is being repainted, in editor coordinates. To get the coordinates for *dc*, offset editor coordinates by adding (*dx*, *dy*). See section 8.5 (page 152) for information about *draw-caret*.

  See also `invalidate-bitmap-cache`.

`on-save-file`

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

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

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

`own-caret`

Tells the editor to display or not display the selection.

The focus state of an editor can be changed by by the system, and such changes do not go through this method; use `on-focus` to monitor focus changes.

- (send *an-editor* `own-caret` *own?*) ⇒ void
    own? : boolean

  Propagates the flag to any snip with the editor-local focus. If no sub-subffers are active, the editor assumes the caret ownership.

paste

Pastes the current contents of the clipboard into the editor.

The system may execute a paste (in response to other method calls) without calling this method. To extend or re-implement copying, override the `do-paste` in `text%` or `do-paste` in `pasteboard%` method of an editor.

See also `get-paste-text-only` .

- (send *an-editor* paste *time*) ⇒ void
   time = 0 : exact integer

  See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

print

Prints the editor.

- (send *an-editor* print *interactive? fit-on-page? output-mode parent*) ⇒ void
   interactive? = #t : boolean
   fit-on-page? = #t : boolean
   output-mode = 'standard : symbol in '(standard postscript)
   parent = #f : frame% or dialog% object or #f

  If *interactive?* is true and a PostScript file is created, the is given a dialog for adjusting printing parameters; see also `get-ps-setup-from-user`. Otherwise, if a PostScript file is created, the settings returned by `current-ps-setup` are used. (The user may still get a dialog to select an output file name; see `post-script-dc%` for more details.

  If *fit-on-page?* is a true value, then during printing for a `text%` editor, the editor's maximum width is set to the width of the page (less margins) and the autowrapping bitmap is removed.

  The *output-mode* setting is used for Windows and MacOS. It determines whether the output is generated directly as a PostScript file (using MrEd's built-in PostScript system) or generated using the platform-specific standard printing mechanism. The possible values as

  – 'standard — print using the platform-standard mechanism (via a `printer-dc%`)
  – 'postscript — print to a PostScript file (via a `post-script-dc%`)

  If *parent* is not #f, it is used as the parent window for configuration dialogs (for either PostScript or platform-standard printing). If *parent* is #f and if the editor is displayed in a single canvas, then the canvas's top-level frame is used as the parent for configuration dialogs. Otherwise, configuration dialogs will have no parent.

  The printing margins are determined by `get-editor-margin` in the current `ps-setup%` object (as determined by `current-ps-setup`).

  For properly spaced text in PostScript output, make sure that the AFM directory is correct in `set-afm-path`.

print-to-dc

Prints the editor into the given drawing context. See also `print` .

- (send *an-editor* print-to-dc *dc*) ⇒ void
   dc : dc<%> object

`put-file`

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

- (send *an-editor* `put-file` *directory* *default-name*) $\Rightarrow$ string or `#f`
  directory : string or `#f`
  default-name : string or `#f`

  Calls the global `put-file`procedure.

  If the editor is displayed in a single canvas, then the canvas's top-level frame is used as the parent for the file dialog. Otherwise, the file dialog will have no parent.

`read-footer-from-file`

See `read-header-from-file`.

- (send *an-editor* `read-footer-from-file` *stream* *name*) $\Rightarrow$ boolean
  stream : `editor-stream-in%` object
  name : string

`read-from-file`

Reads new contents for the editor from a stream. The return value is `#t` if there are no errors, `#f` otherwise. See also section 8.2 (page 150).

The stream provides either new mappings for names in the editor's style list, or it indicates that the editor should share a previously-read style list (depending on how style lists were shared when the editor was written to the stream; see also `write-to-file`).

- In the former case, if the *overwrite-styles?* argument is is `#f`, then each style name in the loaded file that is already in the current style list keeps its current style. Otherwise, existing named styles are overwritten with specifications from the loaded file.

- In the latter case, the editor's style list will be changed to the previously-read list.

- (send *an-editor* `read-from-file` *stream* *overwrite-styles?*) $\Rightarrow$ boolean
  stream : `editor-stream-in%` object
  overwrite-styles? = `#t` : boolean

`read-header-from-file`

Called to handle a named header that is found when reading editor data from a stream. The return value is `#t` if there are no errors, `#f` otherwise.

Override this method only to embellish the file format with new header information. Always call the inherited method if the derived reader does not recognize the header.

- (send *an-editor* `read-header-from-file` *stream* *name*) $\Rightarrow$ boolean
  stream : `editor-stream-in%` object
  name : string

  See also section 8.2 (page 150).

`redo`

Undoes the last undo, if no other changes have been made since.

The system may perform a redo without calling this method in response to other method calls. Use methods such as `on-change` to monitor editor content changes.

See also `add-undo` .

- (send *an-editor* redo) ⇒ void

    If the editor is currently perfoming an undo or redo, the method call is ignored.

`refresh`

Repaints a region of the editor, generally called by an editor administrator.

- (send *an-editor* refresh *x y width height draw-caret*) ⇒ void
    x : real number
    y : real number
    width : non-negative real number
    height : non-negative real number
    draw-caret : symbol in '(no-caret show-inactive-caret show-caret)

    The *x*, *y*, *width*, and *height* arguments specify the area that needs repainting in editor coordinates. See section 8.5 (page 152) for information about *draw-caret*.

`refresh-delayed?`

Returns `#t` if updating on this editor is currently delayed. Updating may be delayed because `begin-edit-sequence` has been called for this editor, or because the editor has no administrator, or because the editor's administrator returns `#t` from its `refresh-delayed?` method. (The administator might return `#t` because an enclosing editor's refresh is delayed.)

- (send *an-editor* refresh-delayed?) ⇒ boolean

`release-snip`

Requests that the specified snip be deleted and released from the editor. If this editor is not the snip's owner or if the snip cannot be released, then `#f` is returned. Otherwise, `#t` is returned and the snip is no longer owned.

See also `release-snip` in `snip-admin%` .

- (send *an-editor* release-snip *snip*) ⇒ void
    snip : `snip%` object

`remove-canvas`

Removes a canvas from this editor's list of displaying canvases. (See `get-canvases`.)

Normally, this method is called only by `set-editor` in `editor-canvas%` .

- (send *an-editor* `remove-canvas` *canvas*) ⇒ void
  canvas : `editor-canvas%` object

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-editor* `resized` *snip redraw-now?*) ⇒ void
  snip : `snip%` object
  redraw-now? : boolean

  If *redraw-now?* is `#f`, the editor will require another message to repaint itself. (See also `needs-update`.)

save-file

Saves the editor into a file, returning `#t` if sucessful, `#f` otherwise.

The filename and format used to save the file can be retrieved with `get-filename`. In a `text%` instance, the format can be retrieved with `get-file-format`.

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

- (send *an-editor* `save-file` *filename format show-errors?*) ⇒ boolean
  filename = `#f` : string or `#f`
  format = `'same` : symbol in `'(guess standard text text-force-cr same copy)`
  show-errors? = `#t` : boolean

  If *filename* is `#f`, then the internally stored filename will be used; if *filename* is `""` or if the internal name is unset or temporary, then the user will be prompted for a name. The possible values for *format* are described at `load-file`. If *show-errors?* is `#f`, then error messages reporting save errors (printed to stdout) are suppressed.

scroll-line-location

Maps a vertical scroll position to a vertical location within the editor.

For `text%` objects: Calling this method may force the recalculation of location information, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

- (send *an-editor* `scroll-line-location` *pos*) ⇒ non-negative real number
  pos : exact integer

scroll-to

Called (indirectly) by snips within the editor: it causes the editor to be scrolled so that a given location range within a given snip is visible. If the editor is scrolled, `#t` is returned, otherwise `#f` is returned.

- (send *an-editor* `scroll-to` *snip localx localy width height refresh? bias*) ⇒ boolean
  snip : `snip%` object
  localx : real number

> *localy* : real number
> *width* : non-negative real number
> *height* : non-negative real number
> *refresh?* : boolean
> *bias* = Symbolnone : symbol in '(start end none)

The *localx*, *localy*, *width*, and *height* arguments specify the area that needs to be viisble in *snip*'s coordinate system.

When the specified region cannot fit in the visible area, *bias* indicates which end of the region to display. When *bias* is 'start, then the top-left of the region is displayed. When *bias* is 'end, then the bottom-right of the region is displayed. Otherwise, *bias* must be 'none.

select-all

Selects all data in the editor

- (send *an-editor* select-all) ⇒ void

set-active-canvas

Sets the active canvas for this editor. (See `get-active-canvas`.)

Normally, this method is called only by `on-focus` in `editor-canvas%` in an editor canvas that is displaying an editor.

- (send *an-editor* set-active-canvas *canvas*) ⇒ void
    canvas : editor-canvas% object

set-admin

Sets the editor's administrator. This method is only called by an administrator.

The administrator of an editor can be changed by by the system, and such changes do not go through this method. A program cannot detect when the administrator changes except by polling `get-admin`.

- (send *an-editor* set-admin *admin*) ⇒ void
    admin : editor-admin% object or #f

set-caret-owner

Sets the keyboard focus owner within an editor or globally.

If #f is provided as the new owner, then the local focus is moved to the editor itself. Otherwise, the local focus is moved to the specified snip.

The domain of focus-setting is one of:

- 'immediate — only set the focus owner within the editor

- 'display — make this editor or the new focus owner get the keyboard focus among the editors in this editor's display (if this is an embedded editor)

- `'global` — make this editor or the new focus owner get the keyboard focus among all elements in the editor's frame

The focus state of an editor can be changed by by the system, and such changes do not go through this method; use `on-focus` to monitor focus changes.

See also `get-focus-snip` .

- (send *an-editor* `set-caret-owner` *snip domain*) ⇒ void
  snip : `snip%` object or `#f`
  domain = `'immediate` : symbol in `'(immediate display global)`

Attempts to give the keyboard focus to *snip*. If *snip* is `#f`, then the caret is taken away from any snip in the editor that currently has the caret and restored to this editor.

If the keyboard focus is moved to *snip* and the editor has the real keyboard focus, the `own-caret` method of the snip will be called.

set-cursor

Sets a custom cursor for the editor. If the custom cursor is `#f`, the current cursor is removed, and a cursor is selected automatically by the editor (depending on whether the cursor is pointing at a clickback). See `adjust-cursor` for more information about the default selection.

An embedding editor's custom cursor can override the cursor of an embedded editor — even if the embedded editor has the caret — if the cursor is specified as an overriding cursor.

- (send *an-editor* `set-cursor` *cursor override?*) ⇒ void
  cursor : `cursor%` object or `#f`
  override? = `#t` : boolean

Sets the custom cursor for the editor to *cursor*. If *override?* is a true value and *cursor* is not `#f`, then this cursor overrides cursor settings in embedded editors.

set-filename

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

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

- (send *an-editor* `set-filename` *filename temporary?*) ⇒ void
  filename : string or `#f`
  temporary? = `#f` : boolean

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

set-inactive-caret-threshold

Sets the threshold for painting an inactive selection. See `get-inactive-caret-threshold` for more information.

    - (send *an-editor* `set-inactive-caret-threshold` *threshold*) ⇒ void
      threshold : symbol in '(`no-caret show-inactive-caret show-caret`)

set-keymap

Sets the current keymap for the editor. A `#f` argument removes all key mapping.

    - (send *an-editor* `set-keymap` *keymap*) ⇒ void
      keymap = `#f` : `keymap%` object or `#f`

set-load-overwrites-styles

Determines whether named styles in the current style list are replaced by `load-file` when the loaded file contains style specifications.

See also `get-load-overwrites-styles` and `read-from-file`.

    - (send *an-editor* `set-load-overwrites-styles` *overwrite?*) ⇒ void
      overwrite? : boolean

set-max-height

Sets the maximum display height for the contents of the editor. A value less or equal to `0` indicates that there is no maximum.

Setting the height is disallowed when the editor is internally locked for reflowing (see also section 8.8 (page 153)).

    - (send *an-editor* `set-max-height` *width*) ⇒ void
      width : non-negative real number or '`none`

set-max-undo-history

Sets the maximum number of undoables that will be remembered by the editor.

    - (send *an-editor* `set-max-undo-history` *count*) ⇒ void
      count : exact integer in [0, 100000]

set-max-width

Sets the maximum display width for the contents of the editor; zero or '`none`' indicates that there is no maximum. In a text editor, having no maximum disables automatic line breaking, and the minimum (positive) maximum width depends on the width of the autowrap bitmap.

Setting the width is disallowed when the editor is internally locked for reflowing (see also section 8.8 (page 153)).

See also `set-autowrap-bitmap`.

    - (send *an-editor* `set-max-width` *width*) ⇒ void
      width : non-negative real number or '`none`

`set-min-height`

Sets the minimum display height for the contents of the editor; zero or `'none` indicates that there is no minimum.

Setting the height is disallowed when the editor is internally locked for reflowing (see also section 8.8 (page 153)).

- (send *an-editor* `set-min-height` *width*) ⇒ void
    width : non-negative real number or `'none`

`set-min-width`

Sets the minimum display width for the contents of the editor; zero or `'none` indicates that there is no minimum.

Setting the width is disallowed when the editor is internally locked for reflowing (see also section 8.8 (page 153)).

- (send *an-editor* `set-min-width` *width*) ⇒ void
    width : non-negative real number or `'none`

`set-modified`

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

See also `is-modified?`.

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

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

`set-paste-text-only`

Sets whether the editor accepts only text from the clipboard, or both text and snips. By default, an editor accepts both text and snips.

See also `get-paste-text-only` .

- (send *an-editor* `set-paste-text-only` *text-only?*) ⇒ void
    text-only? : boolean

`set-snip-data`

Sets extra data associated with the snip (e.g., location information in a pasteboard). See section 8.2.1 (page 151) for more information.

- (send *an-editor* `set-snip-data` *thesnip data*) ⇒ void
    thesnip : `snip%` object
    data : `editor-data%` object

`set-style-list`

Sets the editor's style list. Styles currently in use with the old style list will be "moved" to the new style list. In this "move", if a named style already exists in the new style list, then the new style with the same name will be used in place of the old style.

Setting the style list is disallowed when the editor is internally locked for reflowing (see also section 8.8 (page 153)).

- (send *an-editor* `set-style-list` *style-list*) ⇒ void
    style-list : `style-list%` object

`size-cache-invalid`

Usually called by the editor's administrator. It notifies that editor that it will have to re-calculate all graphic information before re-displaying itself.

- (send *an-editor* `size-cache-invalid`) ⇒ void

`style-has-changed`

Notifies the editor that a style in its style list has changed. This method is automtaically registered with the editor's style list using `notify-on-change` in `style-list%`and automatically deregistered when the style list is removed from the editor.

See `notify-on-change` in `style-list%` for more information.

- (send *an-editor* `style-has-changed` *style*) ⇒ void
    style : `style<%>` object or `#f`

`undo`

Undoes the last editor change.

The system may perform an undo without calling this method in response to other method calls. Use methods such as `on-change` to monitor editor content changes.

See also `add-undo` .

- (send *an-editor* `undo`) ⇒ void

    If the editor is currently perfoming an undo or redo, the method call is ignored.

`write-footers-to-file`

See `write-headers-to-file`.

- (send *an-editor* `write-footers-to-file` *stream*) ⇒ boolean
    stream : `editor-stream-out%` object

`write-headers-to-file`

Called when the editor is being saved to a file. The return value is `#t` if there are no errors, `#f` otherwise. Override this method to add custom header data to a file, but always call the inherited method so that it can write its own extra headers.

To write a header item, call `begin-write-header-footer-to-file`, passing a box for an integer. Then write the header data and end by calling `end-write-header-footer-to-file`, passing back the integer that was put into the box. Follow this procedure correctly or the file will be corrupted.

- (send *an-editor* `write-headers-to-file` *stream*) ⇒ boolean
    stream : `editor-stream-out%` object

    Does nothing.

`write-to-file`

Writes the current editor contents to the given stream. The return value is `#t` if there are no errors, `#f` otherwise. See also section 8.2 (page 150).

If the editor's style list has already been written to the stream, it is not re-written. Instead, the editor content indicates that the editor shares a previously-written style list. This sharing will be recreated when the stream is later read.

- (send *an-editor* `write-to-file` *stream*) ⇒ boolean
    stream : `editor-stream-out%` object

9.5 `editor-admin%`

See section 8.1.1 (page 148) for information about the role of administrators. The `editor-admin%` class is never instantiated directly. It is not even instantiated through derived classes by most programmers; each `editor-canvas%` and `editor-snip%` object creates its own administrator. However, it may be useful to derive a new instance of this class to display editors in a new context. Also, it may be useful to call the methods of an existing administrator from an owned editor.

To create a new `editor-admin%` class, all methods described here must be overridden. They are all invoked by the administrator's editor.

- (`make-object` `editor-admin%`) ⇒ `editor-admin%` object

    Creates a (useless) editor administrator.

`get-dc`

Returns either the drawing context into which the editor is displayed, or the context into which it is currently being drawn. When the editor is not embedded, the returned context is always the drawing content into which the editor is displayed. If the editor is not displayed, `#f` is returned.

The origin of the drawing context is also returned, translated into the local coordinates of the editor. For

an embedded editor, the returned origin is reliable only while the editor is being drawn, or while it receives a mouse or keyboard event.

- (send *an-editor-admin* `get-dc` *x* *y*) $\Rightarrow$ dc<%> object or #f
    x = #f : boxed real number or #f
    y = #f : boxed real number or #f

  The *x* box is filled with the x-origin of the DC in editor coordinates, unless *x* is #f. The *y* box is filled with the y-origin of the DC in editor coordinates, unless *y* is #f.

  See also `editor-location-to-dc-location` in `editor<%>`and `dc-location-to-editor-location` in `editor<%>`.

`get-max-view`

Same as `get-view` unless the editor is visible in multiple standard displays. If the editor has mulitple displays, a region is computed that includes the visible region in all displays.

- (send *an-editor-admin* `get-max-view` *x* *y* *w* *h* *full?*) $\Rightarrow$ void
    x : boxed real number or #f
    y : boxed real number or #f
    w : boxed non-negative real number or #f
    h : boxed non-negative real number or #f
    full? = #f : boolean

  See `get-view`.

`get-view`

Gets the the visible region of the editor within its display (in editor coordinates), or the overall size of the viewing region in the editor's top-level display (for an embedded editor).

If the display is an editor canvas, see also `reflow-container`. The viewing area within an editor canvas is not the full client area of the canvas, because an editor canvas installs a whitespace border around a displayed editor within the client area.

The calculation of the editor's visible region is based on the current size and scrollbar values of the top-level display. For an editor canvas display, the region reported by `get-view` does not depend on whether the canvas is hidden, obscured by other windows, or moved off the edge of the screen.

- (send *an-editor-admin* `get-view` *x* *y* *w* *h* *full?*) $\Rightarrow$ void
    x : boxed real number or #f
    y : boxed real number or #f
    w : boxed non-negative real number or #f
    h : boxed non-negative real number or #f
    full? = #f : boolean

  The *x* box is filled with the left edge of the visible region in editor coordinates, unless *x* is #f. The *y* box is filled with the top edge of the visible region in editor coordinates, unless *y* is #f. The *w* box is filled with the width of the visible region, which may be larger than the editor itself, unless *w* is #f. The *h* box is filled with the height of the visible region, which may be larger than the editor itself, unless *h* is #f.

  If an editor is fully visible and *full?* is #f, then *x* and *y* will both be filled with 0.

If *full?* is a true value, then the returned area is the view area of the top-level display for the editor. This result is different only when the editor is embedded in another editor; in that case, the x and y values may be meaningless, because they are in the coordinate system of the immediate editor within the top-level display.

grab-caret

Called by the editor to request the keyboard focus. If the request is granted, then the administered editor's `own-caret` method will be called.

- (send *an-editor-admin* grab-caret *domain*) ⇒ void
    domain = 'global : symbol in '(immediate display global)

See `set-caret-owner` for information about the possible values of *domain*.

needs-update

Called by the editor to request a refresh to its displayed representation. When the administrator decides that the displayed should be refreshed, it calls the editor's `refresh` method.

- (send *an-editor-admin* needs-update *localx* *localy* *w* *h*) ⇒ void
    localx : real number
    localy : real number
    w : non-negative real number
    h : non-negative real number

The *localx*, *localy*, *w*, and *h* arguments specify a region of the editor to be updated (in editor coordinates).

popup-menu

Opens a popup menu in the display for this editor. The result is **#t** if the popup succeeds, **#f** otherwise (independent of whether the user selects an item in the popup menu).

While the menu is popped up, its target is set to the top-level editor in this editor's display. See `get-popup-target` for more information.

- (send *an-editor-admin* popup-menu *menu* *x* *y*) ⇒ bool
    menu : popup-menu% object
    x : real number
    y : real number

The menu is displayed at x and y in editor coordinates.

refresh-delayed?

Returns **#t** if updating on this administrator's display is currently delayed (usually by `begin-edit-sequence` in `editor<%>` in an enclosing editor).

- (send *an-editor-admin* refresh-delayed?) ⇒ boolean

`resized`

Called by the editor to notify its display that the editor's size or scroll count has changed, so the scrollbars need to be adjusted to reflect the new size. The editor generally needs to be updated after a resize, but the editor decides whether the update should occur immediately.

- (`send` *an-editor-admin* `resized` *refresh?*) ⇒ void
    refresh? : boolean

    If *refresh?* is not `#f`, then the editor is requesting to be updated immediately.

`scroll-to`

Called by the editor to request scrolling so that the given region is visible. The editor generally needs to be updated after a scroll, but the editor decides whether the update should occur immediately.

- (`send` *an-editor-admin* `scroll-to` *localx* *localy* *w* *h* *refresh?* *bias*) ⇒ boolean
    localx : real number
    localy : real number
    w : non-negative real number
    h : non-negative real number
    refresh? = `#t` : boolean
    bias = `'none` : symbol in `'(start end none)`

    The *localx*, *localy*, *w*, and *h* arguments specify a region of the editor to be made visible by the scroll (in editor coordinates).

    If *refresh?* is not `#f`, then the editor is requesting to be updated immediately.

    The *bias* argument is one of:
    - `'start` — if the range doesn't fit in the visible area, show the top-left region
    - `'none` — no special scrolling instructions
    - `'end` — if the range doesn't fit in the visible area, show the bottom-right region

`update-cursor`

Queues an update for the cursor in the display for this editor. The actual cursor used will be determined by calling the editor's `adjust-cursor` method.

- (`send` *an-editor-admin* `update-cursor`) ⇒ void

9.6 `editor-canvas%`

Implements: `canvas<%>`

An `editor-canvas%` object manages and displays a `text%` or `pasteboard%` object.

- (`make-object` `editor-canvas%` *parent* *editor* *style* *scrolls-per-page*) ⇒ `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-hscroll no-vscroll hide-hscroll hide-vscroll)`
    scrolls-per-page = 100 : exact integer in [1, 10000]

The *style* list can contain the following flags:

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

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

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

allow-scroll-to-last

Enbales or disables last-line scrolling, or gets the current enable state. If last-line scrolling is enabled, then an editor displayed in this canvas can be scrolled so that the last line of text is at the top of the canvas (or bottom of the canvas when bottom-based scrolling is enabled; see `scroll-with-bottom-base`). By default, an editor can only be scrolled until the last line is at the bottom (or top) of the canvas.

  - (send *an-editor-canvas* `allow-scroll-to-last`) ⇒ boolean

    Returns `#t` if last-line scrolling is enabled, `#f` otherwise.

  - (send *an-editor-canvas* `allow-scroll-to-last` *on?*) ⇒ void
      on? : boolean

    If *on?* is `#f`, last-line scrolling is disabled, otherwise it is enabled.

allow-tab-exit

Gets or sets whether tab-exit is enabled for the editor canvas. When tab-exit is enabled, the user can move the keyboard focus out of the editor using the Tab and arrow keys, or invoke the default button using the Enter/Return key. By default, tab-exit is disabled.

When tab-exit is enabled for an editor canvas, Tab, arrow, and Enter keyboard events are consumed by a frame's default `on-traverse-char` method. (In addition, a dialog's default method consumes Escape key events.) Otherwise, `on-traverse-char` allows the keyboard events to be propagated to the canvas.

  - (send *an-editor-canvas* `allow-tab-exit`) ⇒ boolean

    Returns `#t` if tab-exit is enabled for the canvas, `#f` otherwise.

  - (send *an-editor-canvas* `allow-tab-exit` *on?*) ⇒ void
      on? : boolean

    Enables or disables tab-exit for the canvas.

call-as-primary-owner

Calls a thunk and returns the value. While the thunk is being called, if the canvas has an editor, the editors's `get-admin` method returns the administrator for this canvas. This method is only useful when an editor is displayed in multiple canvases.

  - (send *an-editor-canvas* `call-as-primary-owner` *f*) ⇒ return value of *f*
      f : procedure of zero arguments

    Returns (*f*).

`force-display-focus`

Enables or disables force-focus mode. In force-focus mode, the caret of the editor displayed in this canvas will always be visible, even when the canvas does not actually have the keyboard focus.

- (send *an-editor-canvas* `force-display-focus`) $\Rightarrow$ boolean

  Returns `#t` if force-focus mode is enabled, `#f` otherwise.

- (send *an-editor-canvas* `force-display-focus` *on?*) $\Rightarrow$ void
    on? : boolean

  If *on?* is `#f`, the focus is displayed normally, otherwise the focus display is forced.

`get-editor`

Returns the editor currently displayed by this canvas, or `#f` if the canvas does not have an editor.

- (send *an-editor-canvas* `get-editor`) $\Rightarrow$ `text%` or `pasteboard%` object or `#f`

`lazy-refresh`

Enables or disables lazy-refresh mode, or gets the current enable state. In lazy-refresh mode, the canvas's `refresh` method is called when the window needs to be updated, rather than `on-paint`. By default, an `editor-canvas%` object is *not* in lazy-refresh mode.

- (send *an-editor-canvas* `lazy-refresh`) $\Rightarrow$ boolean

  Returns `#t` if lazy-refresh mode is enabled, `#f` otherwise.

- (send *an-editor-canvas* `lazy-refresh` *on?*) $\Rightarrow$ void
    on? : boolean

  If *on?* if `#f`, lazy-refresh mode is disabled, otherwise it is enabled.

`on-char`

Called when the canvas receives a keyboard event. See also section 2.3 (page 12).

- (send *an-editor-canvas* `on-char` *event*) $\Rightarrow$ void
    event : `key-event%` object

  Passes the event to the canvas's editor, if any, by calling its `on-char` method.

  See also `get-editor` .

`on-event`

Called when the canvas receives a mouse event. See also section 2.3 (page 12).

- (send *an-editor-canvas* `on-event` *event*) $\Rightarrow$ void
    event : `mouse-event%` object

  Passes the event to the canvas's editor, if any, by calling its `on-char` method.

  See also `get-editor` .

on-focus

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

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

- (send *an-editor-canvas* `on-focus` *on?*) $\Rightarrow$ void
    on? : boolean

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

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 *an-editor-canvas* `on-paint`) $\Rightarrow$ void

  Repaints the editor.

on-scroll

Called when the user changes one of the canvas's manual scrollbars. A `scroll-event%` argument provides information about the scroll action.

This method is not called when automatic scrollbars are changed; the `on-paint` method is called instead.

- (send *an-editor-canvas* `on-scroll` *event*) $\Rightarrow$ void
    event : `scroll-event%` object

  Repaints the editor.

on-size

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

- (send *an-editor-canvas* `on-size` *width* *height*) $\Rightarrow$ void
    width : exact integer in [0, 10000]
    height : exact integer in [0, 10000]

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

scroll-with-bottom-base

Enables or disables bottom-base scrolling, or gets the current enable state. If bottom-base scrolling is on, then scroll positions are determined by line boundaries aligned with the bottom of the viewable area (rather than with the top of the viewable area). If last-line scrolling is also enabled (see `allow-scroll-to-last`), then the editor is bottom-aligned in the display area even when the editor does not fill the viewable area.

- (send *an-editor-canvas* `scroll-with-bottom-base`) ⇒ boolean

  Returns `#t` if bottom-based scrolling is enabled, `#f` otherwise.

- (send *an-editor-canvas* `scroll-with-bottom-base` *on?*) ⇒ void
    on? : boolean

  If *on?* is `#f`, bottom-based scrolling is disabled, otherwise it is enabled.

`set-editor`

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

- (send *an-editor-canvas* `set-editor` *edit redraw?*) ⇒ void
    edit : `text%` or `pasteboard%` object or `#f`
    redraw? = `#t` : boolean

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

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

`set-line-count`

Sets the canvas's graphical minimum height to display a particular number of lines of text. The line height is determined by measuring the difference between the top and bottom of a displayed editor's first line. The minimum height is not changed until the canvas gets an editor. When the canvas's editor is changed, the minimum height is recalculated.

If the line count is set to `#f`, then the canvas's graphical minimum height is restored to its original value.

- (send *an-editor-canvas* `set-line-count` *count*) ⇒ void
    count : exact integer in [1, 1000] or `#f`

9.7 `editor-data%`

An `editor-data%` object contains extra data associated to a snip or region in an editor. See also section 8.2.1 (page 151).

- (`make-object` `editor-data%`) ⇒ `editor-data%` object

  The element returned by `get-next` is initialized to `#f`.

`get-dataclass`

Gets the class for this data.

- (send *an-editor-data* `get-dataclass`) ⇒ `editor-data-class%` object or `#f`

get-next

Gets the next editor data element in a list of editor data elements. A **#f** terminates the list.

- (send *an-editor-data* get-next) ⇒ editor-data% object or #f

set-dataclass

Sets the class for this data.

- (send *an-editor-data* set-dataclass *v*) ⇒ void
    v : editor-data-class% object

set-next

Sets the next editor data element in a list of editor data elements. A **#f** terminates the list.

- (send *an-editor-data* set-next *v*) ⇒ void
    v : editor-data% object or #f

write

Writes the data to the specified stream, returing **#t** if data is written successfully or **#f** otherwise.

- (send *an-editor-data* write *f*) ⇒ boolean
    f : editor-stream-out% object

9.8   `editor-data-class%`

An `editor-data-class%` object defines a type for `editor-data%` objects. See also section 8.2.1 (page 151).

- (make-object editor-data-class%) ⇒ editor-data-class% object

get-classname

Gets the name of the class. Names starting with "wx" are reserved for internal use.

- (send *an-editor-data-class* get-classname) ⇒ string

read

Reads a new data object from the given stream, returning **#f** if there is an error.

- (send *an-editor-data-class* read *f*) ⇒ editor-data% object or #f
    f : editor-stream-in% object

`set-classname`

Sets the name of the class. Names starting with "wx" are reserved for internal use.

- (send *an-editor-data-class* `set-classname` *v*) $\Rightarrow$ void
    v : string

9.9   `editor-data-class-list<%>`

Each eventspace has an instance of `editor-data-class-list<%>`, obtained with (`get-the-editor-data-class-list`). New instances cannot be created directly. This list keeps a list of editor data classes; this list is needed for loading snips from a file. See also section 8.2.1 (page 151).

`add`

Adds a snip data class to the list. If a class with the same name already exists in the list, this one will not be added.

- (send *an-editor-data-class-list* `add` *snipclass*) $\Rightarrow$ void
    snipclass : `editor-data-class%` object

`find`

Finds a snip data class from the list with the given name, returning `#f` if none can be found.

- (send *an-editor-data-class-list* `find` *name*) $\Rightarrow$ `snip-class%` object or `#f`
    name : string

`find-position`

Returns an index into the list for the specified class.

- (send *an-editor-data-class-list* `find-position` *class*) $\Rightarrow$ exact non-negative integer
    class : `editor-data-class%` object

`nth`

Returns the *n*th class in the list (counting from 0), returning `#f` if the list has *n* or less classes.

- (send *an-editor-data-class-list* `nth` *n*) $\Rightarrow$ `editor-data-class%` object or `#f`
    n : exact non-negative integer

`number`

Returns the number of editor data classes in the list.

- (send *an-editor-data-class-list* `number`) $\Rightarrow$ exact non-negative integer

9.10 `editor-snip%`

Superclass: `snip%`

An `editor-snip%` object is a `snip%` object that contains and displays a `editor` object. This snip class is used to insert an editor as a single item within another editor.

- (`make-object 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*) $\Rightarrow$ `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.

`adjust-cursor`

Called to determine the cursor image used when the cursor is moved over the snip in an editor. If `#f` is returned, a default cursor is selected by the editor. (See `adjust-cursor` in `editor<%>` for more information.)

- (`send` *an-editor-snip* `adjust-cursor` *dc x y editorx editory event*) $\Rightarrow$ `cursor%` object or `#f`
  dc : `dc<%>` object
  x : real number
  y : real number
  editorx : real number
  editory : real number
  event : `mouse-event%` object

  Gets a cursor from the embedded editor by calling its `adjust-cursor` method.

`border-visible?`

Returns `#t` if the snip has a border draw adound it, `#f` otherwise.

- (`send` *an-editor-snip* `border-visible?`) $\Rightarrow$ boolean

`get-align-top-line`

Reports whether the snip is in align-top-line mode. See `get-extent` for more information.

See also `set-align-top-line`.

- (send *an-editor-snip* `get-align-top-line`) ⇒ boolean

`get-editor`

Returns the editor contained by the snip or `#f` is there is no editor.

- (send *an-editor-snip* `get-editor`) ⇒ `text%` or `pasteboard%` object or `#f`

`get-extent`

Calculates the snip's width, height, descent (amount of height which is drawn below the baseline), space (amount of height which is "filler" space at the top), and horizontal spaces (amount of width which is "filler" space at the left and right).

A drawing context is provided for the purpose of finding font sizes, but no drawing should occur. The `get-extent` and `partial-offset` methods must not make any assumptions about the state of the drawing context, except that it is scaled properly. In particular, the font for the snip's style is not automatically set in the drawing context before the method is called.[1] If `get-extent` or `partial-offset` changes the drawing context's setting, it must restore them before returning. However, the methods should not need to change the drawing context; only font settings can affect measurement results from a device context, and `get-text-extent` in `dc<%>` accepts a `font%` argument for sizing that overrides that device context's current font.

The snip's left and top locations are provided in editor corrdinates. In a text editor, the y-coordinate is the *line's* top location; the snip's actual top location is potentially undetermined until its height is known.

This method is called by the snip's administrator; it should not be called directly by others. To get the extent of a snip, use `get-snip-location` in `editor<%>` .

The snip's editor is usually internally locked for writing and reflowing when this method is called (see also section 8.8 (page 153)).

- (send *an-editor-snip* `get-extent` *dc x y w h descent space lspace rspace*) ⇒ void
    dc : `dc<%>` object
    x : real number
    y : real number
    w = `#f` : boxed non-negative real number or `#f`
    h = `#f` : boxed non-negative real number or `#f`
    descent = `#f` : boxed non-negative real number or `#f`
    space = `#f` : boxed non-negative real number or `#f`
    lspace = `#f` : boxed non-negative real number or `#f`
    rspace = `#f` : boxed non-negative real number or `#f`

  Calls its editor's `get-extent` method, then adds the editor snip's margins.

  The top space always corresponds to the space of the editor's top line, plus the snip's top margin. Normally, the descent corresponds to the descent of the editor's last line plus the snip's bottom margin.

---

[1]Many snips cache their size information, so automatically setting the font would be wasteful.

However, if the snip is in align-top-line mode (see `set-align-top-line`), the descent correponds to the descent of the top line, plus the height rest of the editor's lines, plus the snip's bottom margin.

If the editor is a text editor, then 1 is normally subtracted from the editor's width as returned by `get-extent`, because the result looks better for editing. If the snip is in tight-text-fit mode (see `set-tight-text-fit`) then 2 is subtracted from a text editor's width, eliminating the two pixels that the text editor reserves for the blinking caret. In addition, tight-text-fit mode subtracts an amount equal to the line spacing from the editor's height. By default, tight-text-fit mode is disabled.

get-inset

Gets the current border insets for the snip. The inset sets how much space is left between the edge of the snip and the border.

- (send *an-editor-snip* `get-inset` *l t r b*) $\Rightarrow$ void
    - *l* : boxed exact non-negative integer
    - *t* : boxed exact non-negative integer
    - *r* : boxed exact non-negative integer
    - *b* : boxed exact non-negative integer

The *l* box is filled with left inset. The *t* box is filled with top inset. The *r* box is filled with right inset. The *b* box is filled with bottom inset.

get-margin

Gets the current margins for the snip. The margin sets how much space is left between the edge of the editor's contents and the edge of the snip.

- (send *an-editor-snip* `get-margin` *l t r b*) $\Rightarrow$ void
    - *l* : boxed exact non-negative integer
    - *t* : boxed exact non-negative integer
    - *r* : boxed exact non-negative integer
    - *b* : boxed exact non-negative integer

The *l* box is filled with left margin. The *t* box is filled with top margin. The *r* box is filled with right margin. The *b* box is filled with bottom margin.

get-max-height

Gets the maximum display height of the snip; zero or `'none` indicates that there is no maximum.

- (send *an-editor-snip* `get-max-height`) $\Rightarrow$ non-negative real number or `'none`

get-max-width

Gets the maximum display width of the snip; zero or `'none` indicates that there is no maximum.

- (send *an-editor-snip* `get-max-width`) $\Rightarrow$ non-negative real number or `'none`

get-min-height

Gets the minimum display height of the snip; zero or `'none` indicates that there is no minimum.

- (send *an-editor-snip* get-min-height) ⇒ non-negative real number or 'none

get-min-width

Gets the minimum display width of the snip; zero or `'none` indicates that there is no minimum.

- (send *an-editor-snip* get-min-width) ⇒ non-negative real number or 'none

get-tight-text-fit

Reports whether the snip is in tight-text-fit mode. See `get-extent` for more information.

See also `set-tight-text-fit`.

- (send *an-editor-snip* get-tight-text-fit) ⇒ boolean

resize

Resizes the snip. The snip can refuse to be resized by returning `#f`. Otherwise, the snip will resize (it must call its administrator's `resized` method) and return `#t`.

See also `on-interactive-resize` in `pasteboard%`.

- (send *an-editor-snip* resize *w* *h*) ⇒ boolean
    w : non-negative real number
    h : non-negative real number

  Also sets the minimum and maximum width of the editor owned by the snip to the given width (minus the snip border space).

set-align-top-line

Enables or disables align-top-line mode. See `get-extent` for more information.

See also `get-align-top-line`.

- (send *an-editor-snip* set-align-top-line *tight?*) ⇒ void
    tight? : boolean

set-editor

Sets the editor contained by the snip, relasing the old editor in the snip (if any). If the new editor already has an administrator, then the new editor is *not* installed into the snip.

When a `editor-snip%` object is not inserted in an editor, it does not have an administrator. During this time, it does not give its contained editor an administrator, either. The administratorless contained editor can therefore "defect" to some other display with an administrator. When a contained editor defects and the snip is eventually inserted into a different editor, the snip drops the traitor contained editor, setting its contained editor to `#f`.

- (send *an-editor-snip* set-editor *editor*) ⇒ void
    editor : text% or pasteboard% object or #f

set-inset

Sets the current border insets for the snip. The inset sets how much space is left between the edge of the snip and the border.

- (send *an-editor-snip* set-inset l t r b) ⇒ void
    - l : exact non-negative integer
    - t : exact non-negative integer
    - r : exact non-negative integer
    - b : exact non-negative integer

set-margin

Sets the current margins for the snip. The margin sets how much space is left between the edge of the editor's contents and the edge of the snip.

- (send *an-editor-snip* set-margin l t r b) ⇒ void
    - l : exact non-negative integer
    - t : exact non-negative integer
    - r : exact non-negative integer
    - b : exact non-negative integer

set-max-height

Sets the maximum display height of the snip; zero or `'none` indicates that there is no maximum.

- (send *an-editor-snip* set-max-height h) ⇒ void
    - h : non-negative real number or `'none`

set-max-width

Sets the maximum display width of the snip; zero or `'none` indicates that there is no maximum.

- (send *an-editor-snip* set-max-width w) ⇒ void
    - w : non-negative real number or `'none`

set-min-height

Sets the minimum display height of the snip; zero or `'none` indicates that there is no minimum.

- (send *an-editor-snip* set-min-height h) ⇒ void
    - h : non-negative real number or `'none`

set-min-width

Sets the minimum display width of the snip; zero or `'none` indicates that there is no minimum.

- (send *an-editor-snip* set-min-width w) ⇒ void
    - w : non-negative real number or `'none`

`set-tight-text-fit`

Enables or disables tight-text-fit mode. See `get-extent` for more information.

See also `get-tight-text-fit`.

- (send *an-editor-snip* `set-tight-text-fit` *tight?*) $\Rightarrow$ void
    tight? : boolean

`show-border`

Shows or hides the snip's border.

- (send *an-editor-snip* `show-border` *show?*) $\Rightarrow$ void
    show? : boolean

  If *show?* is `#f`, the border is hidden, otherwise is it shown.

9.11   `editor-snip-editor-admin<%>`

Extends: (`class->interface editor-admin%`)

An instance of this administrator interface is created with each `editor-snip%` object; new instances cannot be created directly.

`get-snip`

Returns the snip that owns this administrator (and displays the editor controlled by the administrator, if any).

- (send *an-editor-snip-editor-admin* `get-snip`) $\Rightarrow$ `editor-snip%` object

9.12   `editor-stream-in%`

An `editor-stream-in%` object is used to read editor information from a file or other input stream (such as the clipboard).

- (make-object `editor-stream-in%` *base*) $\Rightarrow$ `editor-stream-in%` object
    base : `editor-stream-in-base%` object

  An in-stream base — possible a `editor-stream-in-string-base%` object — must be supplied in *base*.

`>>`

Same as `get`.

- (send *an-editor-stream-in* `>>`*v*) $\Rightarrow$ `editor-stream-in%` object
    v : boxed exact integer

- (send *an-editor-stream-in* `>>`*v*) $\Rightarrow$ `editor-stream-in%` object
    v : boxed real number

`get`

Reads data from the stream, returning itself. Reading from a bad stream always gives 0.

- (send *an-editor-stream-in* `get` *v*) ⇒ `editor-stream-in%` object
    v : boxed exact integer

  The *v* box is filled with the next integer in the stream.

- (send *an-editor-stream-in* `get` *v*) ⇒ `editor-stream-in%` object
    v : boxed real number

  The *v* box is filled with the next floating-point value in the stream.

`get-exact`

Returns the next integer value in the stream.

- (send *an-editor-stream-in* `get-exact`) ⇒ exact integer

`get-fixed`

Gets a fixed-sized integer from the stream. See `put-fixed` for more information. Reading from a bad stream always gives 0.

- (send *an-editor-stream-in* `get-fixed` *v*) ⇒ `editor-stream-in%` object
    v : boxed exact integer

  The *v* box is filled with the fixed-size integer from the stream.

`get-inexact`

Returns the next floating-point value in the stream.

- (send *an-editor-stream-in* `get-inexact`) ⇒ real number

`get-string`

Returns the next string from the stream. Reading from a bad stream returns `#f` or `""`.

- (send *an-editor-stream-in* `get-string` *len*) ⇒ string or `#f`
    len = `#f` : boxed exact non-negative integer or `#f`

  The *len* box is filled with the length of the string, unless *len* is `#f`.

`jump-to`

Jumps to a given position in the stream.

- (send *an-editor-stream-in* `jump-to` *pos*) ⇒ void
    pos : exact non-negative integer

`ok?`

Returns `#t` if the stream is ready for reading, `#f` otherwise. Reading from a bad stream always returns `0` or
`""`.

  - (`send` *an-editor-stream-in* `ok?`) ⇒ boolean

`remove-boundary`

See `set-boundary`.

  - (`send` *an-editor-stream-in* `remove-boundary`) ⇒ void

`set-boundary`

Sets a file-reading boundary at a position in the stream. If there is an attempt to read past this boundary,
an error is signalled. The boundary is removed with a call to `remove-boundary`. Every call to `set-boundary`
must be balanced by a call to `remove-boundary`.

Boundaries help keep a subroutine from reading too much data leading to confusing errors. However, a
malicious subroutine can call `remove-boundary` on its own.

  - (`send` *an-editor-stream-in* `set-boundary` n) ⇒ void
      n : exact non-negative integer

    Sets a file-reading boundary at n bytes past the current stream location.

`skip`

Skips forward in the stream.

  - (`send` *an-editor-stream-in* `skip` n) ⇒ void
      n : exact non-negative integer

    Skips past the next n bytes in the stream.

`tell`

Returns the current stream position.

  - (`send` *an-editor-stream-in* `tell`) ⇒ exact non-negative integer

9.13   `editor-stream-in-base%`

An `editor-stream-in-base%` object is used by a `editor-stream-in%` object to perform low-level reading
of data.

The `editor-stream-in-base%` class is never instantiated directly, but the derived class `editor-stream-in-string-base%`
can be instantiated. New derived classes must override all of the methods decribed in this section.

`bad?`

Returns `#t` if there has been an error reading from the stream, `#f` otherwise.

- (send *an-editor-stream-in-base* bad?) $\Rightarrow$ boolean

`read`

Reads characters to fill the supplied vector. The return value is the number of characters read, which may be less than the number requested if the stream is emptied. If the stream is emptied, the next call to `bad?` must return `#t`.

- (send *an-editor-stream-in-base* read *data*) $\Rightarrow$ exact non-negative integer
    data : vector for characters

`seek`

Moves to the specified absolute position in the stream.

- (send *an-editor-stream-in-base* seek *pos*) $\Rightarrow$ void
    pos : exact non-negative integer

`skip`

Skips over characters in the stream.

- (send *an-editor-stream-in-base* skip *n*) $\Rightarrow$ void
    n : exact non-negative integer

    Skips past the next *n* characters in the stream.

`tell`

Returns the current stream position.

- (send *an-editor-stream-in-base* tell) $\Rightarrow$ exact non-negative integer

9.14   `editor-stream-in-string-base%`

Superclass: `editor-stream-in-base%`

An `editor-stream-in-string-base%` object can be used to read editor data from a string.

- (make-object editor-stream-in-string-base% *s*) $\Rightarrow$ editor-stream-in-string-base% object
    s : string

    Creates a stream base that reads from *s*.

9.15 `editor-stream-out%`

An `editor-stream-out%` object is used to write editor information to a file or other output stream (such as the clipboard).

- (make-object editor-stream-out% *base*) $\Rightarrow$ editor-stream-out% object
    base : editor-stream-out-base% object

  An out-stream base — possibly a `editor-stream-out-string-base%` object — must be supplied in *base*.

`<<`

Same as `put`.

- (send *an-editor-stream-out* <<*v*) $\Rightarrow$ editor-stream-out% object
    v : string

- (send *an-editor-stream-out* <<*v*) $\Rightarrow$ editor-stream-out% object
    v : exact integer

- (send *an-editor-stream-out* <<*v*) $\Rightarrow$ editor-stream-out% object
    v : real number

`jump-to`

Jumps to a given position in the stream.

- (send *an-editor-stream-out* jump-to *pos*) $\Rightarrow$ void
    pos : exact non-negative integer

`ok?`

Returns `#t` if the stream is ready for writing, `#f` otherwise. Writing to a bad stream has no effect.

- (send *an-editor-stream-out* ok?) $\Rightarrow$ boolean

`put`

Writes data to a stream. Writing to a bad stream has no effect.

- (send *an-editor-stream-out* put *n v*) $\Rightarrow$ editor-stream-out% object
    n : exact non-negative integer
    v : string

  Writes *n* characters of the string *v*. The string *v* may contain null characters.

- (send *an-editor-stream-out* put *v*) $\Rightarrow$ editor-stream-out% object
    v : string

  Writes *v*. If *v* has a null character, it will be truncated.

- (send *an-editor-stream-out* `put` *v*) ⇒ `editor-stream-out%` object
    v : exact integer

  Writes an integer.

- (send *an-editor-stream-out* `put` *v*) ⇒ `editor-stream-out%` object
    v : real number

  Writes a floating-point number.

`put-fixed`

Puts a fixed-sized integer into the stream. This method is needed because numbers are usually written in a compressed form (for example, `1` takes one byte, and `512` takes up two bytes, regardless of the C++ type that the number had). In many cases it is useful to temporary write a `0` to a stream, write more data, and then go back and change the `0` to another number; this requires a fixed-size number.

Numbers written to a stream with `put-fixed` must be read with `get-fixed`.

- (send *an-editor-stream-out* `put-fixed` *v*) ⇒ `editor-stream-out%` object
    v : exact integer

`tell`

Returns the current stream position.

- (send *an-editor-stream-out* `tell`) ⇒ exact non-negative integer

9.16   `editor-stream-out-base%`

An `editor-stream-out-base%` object is used by a `editor-stream-out%` object to perform low-level writing of data.

The `editor-stream-out-base%` class is never instantiated directly, but the derived class `editor-stream-out-string-base%` can be instantiated. New derived classes must override all of the methods decribed in this section.

`bad?`

- (send *an-editor-stream-out-base* `bad?`) ⇒ boolean
    Returns `#t` if there has been an error writing to the stream, `#f` otherwise.

`seek`

Moves to the specified absolute position in the stream.

- (send *an-editor-stream-out-base* `seek` *pos*) ⇒ void
    pos : exact non-negative integer

`tell`

Returns the current stream position.

- (send *an-editor-stream-out-base* `tell`) ⇒ exact non-negative integer

write

Writes data to the stream.

- (send *an-editor-stream-out-base* `write` *data*) ⇒ void
    data : list of characters

9.17 `editor-stream-out-string-base%`

Superclass: `editor-stream-out-base%`

An `editor-stream-out-string-base%` object can be used to write editor data into a string.

- (`make-object` `editor-stream-out-string-base%`) ⇒ `editor-stream-out-string-base%` object
    Creates an empty stream.

get-string

Returns the current contents of the stream.

- (send *an-editor-stream-out-string-base* `get-string`) ⇒ string

9.18 `editor-wordbreak-map%`

An `editor-wordbreak-map%` objects is used with a `text%` objects to specify word-breaking criteria for the default wordbreaking function. See also `set-wordbreak-map`, `get-wordbreak-map`, `find-wordbreak`, and `set-wordbreak-func`.

A global object `the-editor-wordbreak-map` is created automatically and used as the default map for all `text` objects.

A workdbreak objects implemnts a mapping from each character to a list of symbols. The following symbols are legal elements of the list:

- `'caret`,
- `'line`,
- `'selection`,
- `'user1`,
- `'user2`

The presence of a flag in a character's value indicates that the character does not break a word when searching for breaks using the corresponding reason. For example, if `'caret` is present, then the character is a non-breaking character for caret-movement words. (Each stream of non-breaking characters is a single word.)

- (make-object editor-wordbreak-map%) ⇒ editor-wordbreak-map% object

  All alpha-numeric characters are initialized with '(caret line selection). All other non-space characters except "-" are initialized with '(line). All space characters and "-" are initialized with null.

get-map

Gets the mapping value for a character. See `editor-wordbreak-map%` for more information.

- (send *an-editor-wordbreak-map* get-map *char*) ⇒ list of symbols
    char : character

  Gets the mapping value for *char*.

set-map

Sets the mapping value for a character. See `editor-wordbreak-map%` for more information.

- (send *an-editor-wordbreak-map* set-map *char value*) ⇒ void
    char : character
    value : list of symbols

  Sets the mapping value for *char* to *value*.

9.19 `image-snip%`

Superclass: `snip%`

An `image-snip%` is a snip that can display bitmap images (usually loaded from a file). When the image file cannot be found, a box containing an "X" is drawn.

- (make-object image-snip% *filename kind relative-path? inline?*) ⇒ image-snip% object
    filename = #f : string or #f
    kind = 'unknown : symbol in '(unknown gif jpeg xbm xpm bmp pict)
    relative-path? = #f : boolean
    inline? = #t : boolean

  Creates an image snip, loading the image *filename* if specified. See also `load-file`.

- (make-object image-snip% *bitmap*) ⇒ image-snip% object
    bitmap : bitmap% object

  Creates an image snip with the given bitmap. See also `set-bitmap`.

get-filename

Returns the name of the currently loaded file, or #f if a file is not loaded (or if a file was loaded with inlining).

- (send *an-image-snip* get-filename *relative-path*) ⇒ string or #f
    relative-path = #f : boxed boolean or #f

  The *relative-path* box is filled with #t if the loaded file's path is relative to the owning editor's path, unless *relative-path* is #f.

`get-filetype`

Returns the kind used to load the currently loaded file, or `'unknown` if a file is not loaded (or if an image was loaded via inlining).

  - (send *an-image-snip* `get-filetype`) ⇒ symbol in `'(unknown gif jpeg xbm xpm bmp pict)`

`load-file`

Loads a new bitmap into the snip.

  - (send *an-image-snip* `load-file` *filename kind relative-path? inline?*) ⇒ void
    filename : string or `#f`
    kind = `'unknown` : symbol in `'(unknown gif jpeg xbm xpm bmp pict)`
    relative-path? = `#f` : boolean
    inline? = `#t` : boolean

    Loads the file by passing *filename* and *kind* to `load-file` If a bitmap had previously been specified with `set-bitmap`, that bitmap will no longer be used. If *filename* is `#f`, then the current image is cleared.

    If *relative-path?* is not `#f` and *filename* is a relative path, then the file will be read using the path of the owning editor's filename. If the image is not inlined, it will be saved as a relative pathname.

    If *inline?* is not `#f`, the image data will be saved directly to the file or clipboard when the image is saved or copied. The source filename is no longer relevant.

`resize`

Resizes the snip. The snip can refuse to be resized by returning `#f`. Otherwise, the snip will resize (it must call its administrator's `resized` method) and return `#t`.

See also `on-interactive-resize` in `pasteboard%`.

  - (send *an-image-snip* `resize` *w h*) ⇒ void
    w : non-negative real number
    h : non-negative real number

    The bitmap will be cropped to fit in the given dimensions.

`set-bitmap`

Sets the bitmap that is displayed by the snip.

A bitmap cannot be used in an `image-snip%` object if it is selected into a `bitmap-dc%` object; if the given bitmap is selected into a `bitmap-dc%` object, an `exn:application:mismatch` exception is raised.

  - (send *an-image-snip* `set-bitmap` *bm*) ⇒ void
    bm : `bitmap%` object

`set-offset`

Sets a graphical offset for the bitmap within the image snip.

- (send *an-image-snip* `set-offset` *dx dy*) ⇒ void
    dx : real number
    dy : real number

9.20   `keymap%`

A `keymap%` object is used by `editor<%>` objects to map keyboard and mouse sequences to arbitrary functions in an extensible way. Keymaps can be used without editors, as well. A `keymap%` object contains

- a mapping from function names to event-handling procedures; and
- a mapping from key and mouse sequences to function names .

A handler procedure in a keymap is invoked with a `key-event%` object or a `mouse-event%` object. It is also given another value that depends on the context in which the keymap is used (or, more specifically, the arguments to `handle-key-event` or `handle-mouse-event`). For keymaps associated with `editor<%>` objects, the extra parameter is generally the `editor<%>` object that received the keyboard or mouse event.

- (`make-object keymap%`) ⇒ `keymap%` object

    Creates an empty keymap.

add-function

Names a new function to handle events, called in response to `handle-key-event`, `handle-key-event`, or `call-function`. The return value is of the procedure is ignored.

If there was already a function mapped to this name, it will be replaced with the given function.

When the function is called, it gets the arguments that were passed to `handle-key-event`, `handle-mouse-event`, or `call-function`. For keymaps asscociated with an editor, this is normally the target editor.

- (send *a-keymap* `add-function` *name func*) ⇒ void
    name : string
    func : procedure of two arguments: an arbitrary value and a `event%` object

break-sequence

Clears the state of the keymap if it is in the middle of a key sequence. For example, the user may have hit escape, and then changed to another window; if escape is part of a keyboard sequence, the keymap state needs to be cleared because the user is not going to complete the sequence.

A break callback function can be installed with `set-break-sequence-callback`.

- (send *a-keymap* `break-sequence`) ⇒ void

call-function

Calls a named event handler directly. If the function cannot be found or the found handler did not want to handle the event, `#f` is returned. Otherwise, the return value is the return value of the event handler.

- (send *a-keymap* `call-function` *name in event try-chain?*) ⇒ boolean
    name : string
    in : value
    event : `event%` object
    try-chain? = `#f` : boolean

The *in* and *event* arguments are passed on to the keymap handler procedure if one is found.

If *try-chain?* is not `#f`, keymaps chained to this one are searched for the function name. If the function is not found and *try-chain?* is `#f`; an exception is also raised, but the exception handler cannot escape (see §2.4.4).

`chain-to-keymap`

Multiple keymaps can be chained off one keymap using `chain-to-keymap`. When keymaps are chained to a main keymap, then events handled by the main keymap are passed to the chained keymaps until a chained keymap handles the events. Keymaps can be chained together in an arbitrary acyclic graph.

Keymap chaining is useful because multiple-event sequences are handled correctly by chained groups. Dispatching each individual event to separate keymaps is problematic withouth chaining because keymaps may acquire state that must be reset when a callback is invoked in one of the keymaps. This state can be maually cleared with `break-sequence`, but this also invokes the handler installed by `set-break-sequence-callback`.

- (send *a-keymap* `chain-to-keymap` *next prefix?*) ⇒ void
    next : `keymap%` object
    prefix? : boolean

If *next* will be used to handle events which are not handled by this keymap. If *prefix?* is a true value, then *next* will take precedence over other keymaps already chained to this one.

`get-double-click-interval`

Returns the maximum number of milliseconds that can separate the clicks of a double-click.

- (send *a-keymap* `get-double-click-interval`) ⇒ exact integer in [0, 1000000]

`handle-key-event`

Attempts to handle a keyboard event, returning `#t` if the event was handled (i.e., a handler was found and it returned a true value), `#f` otherwise.

- (send *a-keymap* `handle-key-event` *in event*) ⇒ boolean
    in : value
    event : `key-event%` object

Seraches for a mapping that matches *event*. See also `call-function`.

`handle-mouse-event`

Attempts to handle a mouse event, returning `#t` if the event was handled (i.e., a handler was found and it returned a true value), `#f` otherwise.

- (send *a-keymap* `handle-mouse-event` *in* *event*) ⇒ boolean
     in : value
     event : `mouse-event%` object

  Seraches for a mapping that matches *event*. See also `call-function`.

`map-function`

Maps an input state to the name of an event handler.

- (send *a-keymap* `map-function` *keyname* *fname*) ⇒ void
     keyname : string
     fname : string

  Maps an input state sequence to a function name using a string-encoded sequence in *keyname*. The format of *keyname* is a sequence of semicolon-delimited input states; each state is made up of a sequence of modifier identifiers followed by a key identifier.

  The modifier identifiers are:

    – "s:" — All platforms: Shift
    – "c:" — All platforms: Control
    – "a:" — MacOS: Option
    – "m:" — Windows: Alt; X: Meta
    – "d:" — MacOS: Command

  If a particular modifier is not mentioned in a state string, it matches states whether that modifier is pressed or not pressed. A tilde ( ) preceding a modifier makes the string match only states where the corresponding modifier is not pressed. If the state string begins with a colon, then the string only matches a state if modifiers not mentioned in the string are not pressed.

  A key identifier can be either a character on the keyboard (e.g., `"a"`, `"2"`, `"?"`) or a special name. The special names are:

    – `"leftbutton"` (button down)
    – `"rightbutton"`
    – `"middlebutton"`
    – `"leftbuttondouble"` (button down for double-click)
    – `"rightbuttondouble"`
    – `"middlebuttondouble"`
    – `"leftbuttontriple"` (button down for triple-click)
    – `"rightbuttontriple"`
    – `"middlebuttontriple"`
    – `"leftbuttonseq"` (all events from button down through button up)
    – `"rightbuttonseq"`
    – `"middlebuttonseq"`
    – `"esc"`
    – `"delete"`
    – `"del"` (same as `"delete"`)
    – `"insert"`
    – `"ins"` (same as `"insert"`)
    – `"add"`
    – `"subtract"`
    – `"multiply"`
    – `"divide"`
    – `"backspace"`
    – `"back"`

- – `"return"`
- – `"enter"` (same as `"return"`)
- – `"tab"`
- – `"space"`
- – `"right"`
- – `"left"`
- – `"up"`
- – `"down"`
- – `"home"`
- – `"end"`
- – `"pageup"`
- – `"pagedown"`
- – `"semicolon"`
- – `"colon"`
- – `"numpad1"`
- – `"numpad2"`
- – `"numpad3"`
- – `"numpad4"`
- – `"numpad5"`
- – `"numpad6"`
- – `"numpad7"`
- – `"numpad8"`
- – `"numpad9"`
- – `"f1"`
- – `"f2"`
- – `"f3"`
- – `"f4"`
- – `"f5"`
- – `"f6"`
- – `"f7"`
- – `"f8"`
- – `"f9"`
- – `"f10"`
- – `"f11"`
- – `"f12"`
- – `"f13"`
- – `"f14"`
- – `"f15"`
- – `"f16"`
- – `"f17"`
- – `"f18"`
- – `"f19"`
- – `"f20"`
- – `"f21"`
- – `"f22"`
- – `"f23"`
- – `"f24"`

For a special keyword, the capitalization does not matter. However, capitalization is important for single-letter keynames (e.g., `"A"` is interpreted as `"s:a"`).

A state can match multiple state strings mapped in a keymap (or keymap chain); when a state matches multiple state strings, a mapping is selected by ranking the strings according to specificity. A state string that mentions more pressed modifiers has a higher rank than other state strings, and if two strings mention the same number of pressed modifiers, the one that metions more unpressed modifiers

has a higher rank. In that case that multiple matching strings have the same rank, one string is selected arbitrarily.

Examples:

- `"space"` — matches whenever the space bar is pressed, regardless of the state of modifiers keys.
- `" c:space"` — matches whenever the space bar is pressed and the Control key is not pressed.
- `"a"` — matches whenever "a" is typed, regardless of the state of modifiers keys other than Shift.
- `":a"` — matches only when "a" is typed with no modifier keys pressed.
- `" c:a"` — matches whenever "a" is typed and neither the Shift key nor the Control key is pressed.
- `":esc;:c:c"` — matches an Escape key press (no modifiers) followed by a Control-C press (no modifiers other than Control).

A call to `map-function` that would map a particular key sequence both as a prefix and as a complete sequence raises an exception, but the exception handler cannot escape (see §2.4.4).

A function name does not have to be mapped to a handler before input states are mapped to the name; the handler is dispatched by name at the time of invocation. The event handler mapped to a function name can be changed without affecting the map from input states to function names.

remove-chained-keymap

Unchains a keymap from this keymap.

- (send *a-keymap* remove-chained-keymap *keymap*) ⇒ void
    keymap : `keymap%` object

If *keymap* was previously chained from this keymap (through `chain-to-keymap`, then it is removed from the chain-to list.

remove-grab-key-function

Removes a callback installed with `set-grab-key-function`.

- (send *a-keymap* remove-grab-key-function) ⇒ void

remove-grab-mouse-function

Removes a callback installed with `set-grab-mouse-function`.

- (send *a-keymap* remove-grab-mouse-function) ⇒ void

set-break-sequence-callback

Installs a callback procedure that is invoked when `break-sequence` is called. After it is invoked once, the callback is removed from the keymap. If another callback is installed before `break-sequence` is called, the old callback is invoked immediately before the new one is installed.

- (send *a-keymap* set-break-sequence-callback *f*) ⇒ void
    f : procedure of no arguments

set-double-click-interval

Sets the maximum number of milliseconds that can separate the clicks of a double-click.

- (send *a-keymap* `set-double-click-interval` *n*) ⇒ void
    n : exact integer in [0, 1000000]

`set-grab-key-function`

Installs a callback procedure that is invoked after the keymap matches input to a function name or fails to match an input. Only one keyboard grab function can be installed at a time. When keymaps are chained to a keymap with a grab callback, the callback is invoked for matches in the chained keymap (when the chained keymap does not have its own grab callback).

If a grab callback returns a true value for a matching or non-matching callback, the event is considered handled. If the callback returns a true value for a matching callback, then the matching keymap function is not called by the keymap.

- (send *a-keymap* `set-grab-key-function` *f*) ⇒ void
    f : procedure of four arguments — a string or `#f`, a `keymap%` object, an arbitrary value, and a `key-event%` object — that returns a boolean
  The callback procedure *f* will be invoked as:

    (*f   str   km   editor   event*)

  The *str* argument is the name of a function for a matching callback, or `#f` for a non-matching callback. The *km* argument is the keymap that matched (possibly a keymap chained to the one in which the callback was installed) or the keymap in which the callback was installed. The *editor* and *event* arguments are the same as passed on to the matching keymap function.

  Key grab callback functions are de-installed with `remove-grab-key-function`.

`set-grab-mouse-function`

Like `set-grab-key-function`, but for mouse events.

- (send *a-keymap* `set-grab-mouse-function` *f*) ⇒ void
    f : procedure of four arguments — a string or `#f`, a `keymap%` object, an arbitrary value, and a `mouse-event%` object — that returns a boolean
  See `set-grab-key-function`.

9.21   `mult-color<%>`

A `mult-color<%>` object is used to scale the RGB values of a `color%` object. A `mult-color<%>` object exist only within a `style-delta%` object.

See also `get-foreground-mult` and `get-background-mult`.

`get`

Gets all of the scaling values.

- (send *a-mult-color* `get` *r   g   b*) ⇒ void
    r : boxed real number
    g : boxed real number
    b : boxed real number

217

The *r* box is filled with the scaling value for the red component of the color. The *g* box is filled with the scaling value for the green component of the color. The *b* box is filled with the scaling value for the blue component of the color.

get-b

Gets the multiplicative scaling value for the blue component of the color.

- (send *a-mult-color* get-b) ⇒ real number

get-g

Gets the multiplicative scaling value for the green component of the color.

- (send *a-mult-color* get-g) ⇒ real number

get-r

Gets the multiplicative scaling value for the red component of the color.

- (send *a-mult-color* get-r) ⇒ real number

set

Sets all of the scaling values.

- (send *a-mult-color* set *r* *g* *b*) ⇒ void
    r : real number
    g : real number
    b : real number

set-b

Sets the multiplicative scaling value for the blue component of the color.

- (send *a-mult-color* set-b *v*) ⇒ void
    v : real number

set-g

Sets the multiplicative scaling value for the green component of the color.

- (send *a-mult-color* set-g *v*) ⇒ void
    v : real number

set-r

Sets the additive value for the red component of the color.

- (send *a-mult-color* `set-r` *v*) ⇒ void
     v : real number

9.22   `pasteboard%`

Implements: `editor<%>`

A `pasteboard%` object is an editor for displaying snips with arbitrary positions.

- (make-object `pasteboard%`) ⇒ `pasteboard%` object
   The editor will not be displayed until it is attached to a `editor-canvas%` object or some other display.
   A new `keymap%` object is created for the new editor. See also `get-keymap` and `set-keymap`.
   A new `style-list` object is created for the new editor. See also `get-style-list` and `set-style-list`.

add-selected

Selects snips without deselecting other snips.

The selection in a pasteboard can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-select` to monitor selection changes.

- (send *a-pasteboard* `add-selected` *snip*) ⇒ void
     snip : `snip%` object

   Selects *snip*.

- (send *a-pasteboard* `add-selected` *x y w h*) ⇒ void
     x : real number
     y : real number
     w : non-negative real number
     h : non-negative real number

   Selects all snips that intersect with the given rectangle (in editor coordinates).

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

after-interactive-move

This method is called after the user stops interactively dragging snips (the ones that are selected; see `find-next-selected-snip`). The mouse event that terminated the move (usually a button-up event) is provided.

See also `can-interactive-move?` and `on-interactive-move`.

- (send *a-pasteboard* `after-interactive-move` *event*) ⇒ void
  event : `mouse-event%` object

  Does nothing.

after-interactive-resize

This method is called after the user stops interactively resizing a snip (the one that is currently selected; see `find-next-selected-snip`).

See also `can-interactive-resize?` and `on-interactive-resize`.

- (send *a-pasteboard* `after-interactive-resize` *snip*) ⇒ void
  snip : `snip%` object

  The *snip* argument is the snip that was resized. This method does nothing.

after-move-to

Called after a given snip is moved within the editor (and after the display is refreshed; use `on-move-to` and `begin-edit-sequence` to avoid extra refreshes when `after-move-to` modifies the editor).

See also `can-move-to?` and `on-edit-sequence`.

No internals locks are set when this method is called.

- (send *a-pasteboard* `after-move-to` *snip x y dragging?*) ⇒ void
  snip : `snip%` object
  x : real number
  y : real number
  dragging? : boolean

  If *dragging?* is not `#f`, then this move was a temporary move for dragging.

`after-resize`

Called after a given snip is resized (and after the display is refreshed; use `on-resize` and `begin-edit-sequence` to avoid extra refreshes when `after-resize` modifies the editor), or after an unsuccessful resize attempt was made

See also `can-resize?` and `on-edit-sequence`.

No internals locks are set when this method is called.

- (send *a-pasteboard* `after-resize` *snip w h resized?*) ⇒ void
    snip : `snip%` object
    w : non-negative real number
    h : non-negative real number
    resized? : boolean
  If *resized?* is not `#f`, the snip was successfully resized.

`after-select`

This method is called after a snip in the pasteboard is selected or deselected. See also `on-select`. This method is not called after selected snip is deleted (and thus de-selected indirectly); see also `after-delete`.

See also `can-select?` and `on-edit-sequence`.

No internals locks are set when this method is called.

- (send *a-pasteboard* `after-select` *snip on?*) ⇒ void
    snip : `snip%` object
    on? : boolean
  If *on?* is `#t`, then *snip* was just selected, otherwise *snip* was just deselected.

`can-delete?`

Called before a snip is deleted from the editor. If the return value is `#f`, then the delete will be aborted.

See also `on-delete` and `after-delete`.

The editor is internally locked for writing when this method is called (see also section 8.8 (page 153)).

- (send *a-pasteboard* `can-delete?` *snip*) ⇒ boolean
    snip : `snip%` object

`can-insert?`

Called before a snip is inserted from the editor. If the return value is `#f`, then the insert will be aborted.

See also `on-insert` and `after-insert`.

The editor is internally locked for writing when this method is called (see also section 8.8 (page 153)).

- (send *a-pasteboard* `can-insert?` *snip before x y*) ⇒ boolean
    snip : `snip%` object

221

> *before* : `snip%` object or `#f`
> *x* : real number
> *y* : real number

can-interactive-move?

This method is called when the user starts interactively dragging snips (the ones that are selected; see `find-next-selected-snip`). All of the selected snips will be moved. If `#f` is returned, the interactive move is disallowed. The mouse event that started the move (usually a button-down event) is provided.

See also `on-interactive-move`, `after-interactive-move`, and `interactive-adjust-move`.

- (send *a-pasteboard* `can-interactive-move?` *event*) ⇒ boolean
  event : `mouse-event%` object

Returns `#t`.

can-interactive-resize?

This method is called when the user starts interactively resizing a snip (the one that is selected; see `find-next-selected-snip`). If `#f` is returned, the interactive resize is disallowed.

See also `after-interactive-resize`, `after-interactive-resize`, and `interactive-adjust-resize`.

- (send *a-pasteboard* `can-interactive-resize?` *snip*) ⇒ boolean
  snip : `snip%` object

The *snip* argument is the snip that will be resized. This method returns `#t`.

can-move-to?

Called before a snip is moved in the editor. If the return value is `#f`, then the move will be aborted.

See also `on-move-to` and `after-move-to`.

The editor is internally locked for writing when this method is called (see also section 8.8 (page 153)).

- (send *a-pasteboard* `can-move-to?` *snip x y dragging?*) ⇒ boolean
  snip : `snip%` object
  x : real number
  y : real number
  dragging? : boolean

If *dragging?* is not `#f`, then this move is a temporary move for dragging.

can-resize?

Called before a snip is resized in the editor. If the return value is `#f`, then the resize will be aborted.

See also `on-resize` and `after-resize`.

The editor is internally locked for writing when this method is called (see also section 8.8 (page 153)).

- (send *a-pasteboard* `can-resize?` *snip w h*) ⇒ boolean
    snip : `snip%` object
    w : non-negative real number
    h : non-negative real number

can-select?

This method is called before a snip in the pasteboard is selected or deselected. If `#f` is returned, the selection change is disallowed. This method is not called when a selected snip is to be deleted (and thus de-selected indirectly); see also `can-delete?` .

See also `on-select` and `after-select`.

The editor is internally locked for writing when this method is called (see also section 8.8 (page 153)).

- (send *a-pasteboard* `can-select?` *snip on?*) ⇒ boolean
    snip : `snip%` object
    on? : boolean

    If *on?* is `#t`, then *snip* will be selected, otherwise *snip* will be deselected.

change-style

Changes the style for items in the editor.

The style within an editor can be changed by the system (in response to other method calls), and such changes do not go through this method; use `on-change-style` in `text%` to monitor style changes.

- (send *a-pasteboard* `change-style` *style snip*) ⇒ void
    style : `style<%>` object
    snip = `#f` : `snip%` object or `#f`

    Changes the style of *snip* by applying a style delta. If *snip* is `#f`, then all currently selected snips are changed.

- (send *a-pasteboard* `change-style` *delta snip*) ⇒ void
    delta : `style-delta%` object
    snip : `snip%` object

    Changes the style of *style* to a specific style. The editor's style list must contain *style*. If *snip* is `#f`, then all currently selected snips are changed.

- (send *a-pasteboard* `change-style` *delta*) ⇒ void
    delta : `style-delta%` object

    Changes the style of the selected items by applying a style delta.

    To change a large collection of snips from one style to another style, consider providing a `style<%>` instance rather than a `style-delta%` instance. Otherwise, `change-style` must convert the `style-delta%` instance to the `style<%>` instance for every snip; this conversion consumes both time and (temporary) memory.

- (send *a-pasteboard* `change-style` *style*) ⇒ void
    style : `style<%>` object

    Changes the style of the selected items to a specific style. The editor's style list must contain *style*, otherwise the style is not changed. See also `convert`.

`copy-self-to`

Copies the properties of this editor into an existing editor.

- (send *a-pasteboard* `copy-self-to` *dest*) ⇒ void
    dest : `text%` or `pasteboard%` object

  Each snip in this editor is copied and inserted into *dest*. In addition, this editor's filename, maximum undo history setting, keymap, interactive caret threshold, and overwrte-styles-on-load settings are installed into *dest*. This editor's style list is copied and the copy is installed as the style list for *dest*.

  This editor's dragability, selection visiblity state, and scroll step are installed into *dest*.

`delete`

The content of an editor can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-delete` to monitor content deletion changes.

- (send *a-pasteboard* `delete`) ⇒ void

  Deletes the currently selected snips from the editor.

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

  Deletes the *snip* from the editor.

`do-copy`

Called to copy the editor's current selection into the clipboard. This method is provided so that it can be overriden by subclasses. Do not call this method directly; instead, call `copy` .

- (send *a-pasteboard* `do-copy` *time* *extend?*) ⇒ void
    time : exact integer
    extend? : boolean

  Copy the current selection, extending the current clipboard contexnts if *extend?* is true.

  See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`do-paste`

Called to paste the current contents of the clipboard into the editor. This method is provided so that it can be overriden by subclasses. Do not call this method directly; instead, call `paste`.

- (send *a-pasteboard* `do-paste` *time*) ⇒ void
    time : exact integer

  See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`erase`

Deletes all snips from the editor.

See also `delete`.

224

- (send *a-pasteboard* `erase`) $\Rightarrow$ void

find-next-selected-snip

Returns a selected snip in the editor.

- (send *a-pasteboard* `find-next-selected-snip` *start*) $\Rightarrow$ snip% object or #f
  start : snip% object or #f

  Returns the next selected snip in the editor, starting the search after *start*. (See section 8.1 (page 148) for information about snip order in pasteboards.) If *start* is #f, then the search starts with the first snip in the editor (and thus returns the first selected snip, if any are selected). If no more selected snips are available, or if *start* is not in the pasteboard, #f is returned.

find-snip

Finds the frontmost snip that intersects with a given location. See section 8.1 (page 148) for information about snip order in pasteboards.

The result is only valid when the editor is displayed (see section 8.1 (page 148)).

- (send *a-pasteboard* `find-snip` *x y*) $\Rightarrow$ snip% object or #f
  x : real number
  y : real number

  The *x* and *y* arguments are in editor corrdinates.

get-center

Returns the center of the pasteboard in pasteboard coordinates.

- (send *a-pasteboard* `get-center` *x y*) $\Rightarrow$ void
  x : boxed real number
  y : boxed real number

  The *x* box is filled with the x-coordinate of the center and *y* is filled with the y-coordinate of the center.

get-dragable

Returns whether snips in the editor can be interactively dragged by event handling in `on-default-event`: #t if dragging is allowed, #f otherwise. By default, dragging is allowed. See also `set-dragable`.

- (send *a-pasteboard* `get-dragable`) $\Rightarrow$ boolean

get-scroll-step

Gets the editor location offset for each vertical scroll position. See also `set-scroll-step`.

- (send *a-pasteboard* `get-scroll-step`) $\Rightarrow$ non-negative real number

`get-selection-visible`

Returns whether selection dots are drawn around the edge of selected snips in the pasteboard. By default, selection dots are on. See also `set-selection-visible`.

- (send *a-pasteboard* `get-selection-visible`) ⇒ boolean

`insert`

Inserts data into the editor.

The content of an editor can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-insert` in `text%` or `on-insert` in `pasteboard%` to monitor content additions changes.

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

   Inserts *snip* at position (*x*, *y*) just in front of *snip*. (See section 8.1 (page 148) for information about snip order in pasteboards.) If *snip* is `#f`, then *snip* is inserted behind all other snips.

- (send *a-pasteboard* `insert` *snip x y*) ⇒ void
   snip : `snip%` object
   x : real number
   y : real number

   Inserts *snip* at position (*x*, *y*) behind all other snips. (See section 8.1 (page 148) for information about snip order in pasteboards.)

- (send *a-pasteboard* `insert` *snip before*) ⇒ void
   snip : `snip%` object
   before : `snip%` object or `#f`

   Inserts *snip* in the center of the editor (with respect to the total width and height of the editor) just in front of *snip*. (See section 8.1 (page 148) for information about snip order in pasteboards.) If *snip* is `#f`, then *snip* is inserted behind all other snips.

- (send *a-pasteboard* `insert` *snip*) ⇒ void
   snip : `snip%` object

   Inserts a snip into the editor. A snip cannot be inserted into multiple editors or multiple times within a single editor.

`interactive-adjust-mouse`

This method is called during interactive dragging and resizing (of the currently selected snips; see `find-next-selected-snip`) to preprocess the current mouse position (in editor coordinates). The snip and actual x and y coordinates are passed into the method (boxed); the resulting coordinates are used instead of the actual mouse position.

See also `interactive-adjust-resize`.

- (send *a-pasteboard* `interactive-adjust-mouse` *x y*) ⇒ void
    x : boxed real number
    y : boxed real number

A negative value for either *x* or *y* is replaced with `0`.

interactive-adjust-move

This method is called during an interactive move (for each selected snip) to preprocess the user-determined snip position for each selected snip. The snip and mouse-determined positions (in editor coodinates) are passed into the method (boxed); the resulting positions are used for graphical feedback to the user during moving.

The actual mouse coordinates are first sent through `interactive-adjust-mouse` before determining the positions passed into this method.

- (send *a-pasteboard* `interactive-adjust-move` *snip x y*) ⇒ void
    snip : `snip%` object
    x : boxed real number
    y : boxed real number

Does nothing.

interactive-adjust-resize

This method is called during interactive resizing of a snip to preprocess the user-determined snip size. The snip and mouse-determined height and width are passed into the method (boxed); the resulting height and width are used for graphical feedback to the user during resizing.

The actual mouse coordinates are first sent through `interactive-adjust-mouse` before determining the sizes passed into this method.

- (send *a-pasteboard* `interactive-adjust-resize` *snip width height*) ⇒ void
    snip : `snip%` object
    width : boxed non-negative real number
    height : boxed non-negative real number

Does nothing.

is-selected?

Returns `#t` if a specified snip is currently selected or `#f` otherwise.

- (send *a-pasteboard* `is-selected?` *snip*) ⇒ boolean
    snip : `snip%` object

lower

Moves the snip one level deeper (i.e., behind one more other snip) in the pasteboard's snip order. See section 8.1 (page 148) for information about snip order in pasteboards.

See also `raise`, `set-before`, and `set-after`.

- (send *a-pasteboard* `lower` *snip*) ⇒ void
    snip : `snip%` object

move

Moves a specified snip a given number of pixels in the horizontal and vertical directions.

Snip locations in a pasteboard can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-move-to` to monitor snip position changes.

- (send *a-pasteboard* `move` *snip* *x* *y*) ⇒ void
    snip : `snip%` object
    x : real number
    y : real number

    Moves *snip* right *x* pixels and down *y* pixels.

- (send *a-pasteboard* `move` *x* *y*) ⇒ void
    x : real number
    y : real number

    Moves all selected snips right *x* pixels and down *y* pixels.

move-to

Moves a specified snip to a given location in the editor.

Snip locations in a pasteboard can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-move-to` to monitor snip position changes.

- (send *a-pasteboard* `move-to` *snip* *x* *y*) ⇒ void
    snip : `snip%` object
    x : real number
    y : real number

no-selected

Deselects all selected snips in the editor.

The selection in a pasteboard can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-select` to monitor selection changes.

- (send *a-pasteboard* `no-selected`) ⇒ void

on-default-event

Called by `on-local-event` when the event is *not* handled by a caret-owning snip or by the keymap.

- (send *a-pasteboard* `on-default-event` *event*) ⇒ void
    event : `mouse-event%` object

    Selects, drags, and resizes snips:

- Clicking on a snip selects the snip. Shift-clicking extends the current selection with the snip.
- Clicking in the space between snips drags a selection box; once the mouse button is released, all snips touching the box are selected. Shift-clicking extends the current selection with the new snips.
- Double-clicking on a snip calls `on-double-click`.
- Clicking on a selected snip drags the selected snip(s) to a new location.
- Clicking on a hilighting tab for a selected object resizes the object.

on-delete

Called before a snip is deleted from the editor, after `can-delete?` is called to verify that the deletion is allowed. The `after-delete` method is guaranteed to be called after the delete has completed.

The editor is internally locked for writing when this method is called (see also section 8.8 (page 153)). Use `after-delete` to modify the editor, if necessary.

- (send *a-pasteboard* on-delete *snip*) $\Rightarrow$ void
    snip : `snip%` object

on-double-click

This method is called when the user double-clicks on a snip in the editor. The clicked-on snip and event records are passed to the method.

- (send *a-pasteboard* on-double-click *snip* *event*) $\Rightarrow$ void
    snip : `snip%` object
    event : `mouse-event%` object

  If *snip* accepts events, it is designated as the caret owner and all snips in the editor are unselected.

on-insert

Called before a snip is inserted from the editor, after `can-insert?` is called to verify that the insertion is allowed. The `after-insert` method is guaranteed to be called after the insert has completed.

The editor is internally locked for writing when this method is called (see also section 8.8 (page 153)). Use `after-insert` to modify the editor, if necessary.

- (send *a-pasteboard* on-insert *snip* *before* *x* *y*) $\Rightarrow$ void
    snip : `snip%` object
    before : `snip%` object or #f
    x : real number
    y : real number

on-interactive-move

This method is called when the user starts interactively dragging snips (the ones that are selected; see `find-next-selected-snip`), after `can-interactive-move?` is called to verify that the move is allowed. The `after-interactive-move` method is guaranteed to be called after the move has completed. All of the selected snips will be moved. The mouse event that started the move (usually a button-down event) is provided.

See also `interactive-adjust-move`.

- (send *a-pasteboard* `on-interactive-move` *event*) ⇒ void
   event : `mouse-event%` object

   Returns `#t`.

on-interactive-resize

This method is called when the user starts interactively resizing a snip (the one that is selected; see `find-next-selected-snip`), after `can-interactive-resize?` is called to verify that the resize is allowed. The `after-interactive-resize` method is guaranteed to be called after the resize has completed.

- (send *a-pasteboard* `on-interactive-resize` *snip*) ⇒ void
   snip : `snip%` object

   The *snip* argument is the snip that will be resized. This method returns `#t`.

on-move-to

Called before a snip is moved in the editor, after `can-move-to?` is called to verify that the move is allowed. The `after-move-to` method is guaranteed to be called after the move has completed.

The editor is internally locked for writing when this method is called (see also section 8.8 (page 153)). Use `after-move-to` to modify the editor, if necessary. See also `on-interactive-move` and `interactive-adjust-move`.

- (send *a-pasteboard* `on-move-to` *snip x y dragging?*) ⇒ void
   snip : `snip%` object
   x : real number
   y : real number
   dragging? : boolean

   If *dragging?* is not `#f`, then this move is a temporary move for dragging.

on-resize

Called before a snip is resized by the editor, after `can-resize?` is called to verify that the resize is allowed. The `after-resize` method is guaranteed to be called after the resize has completed.

The editor is internally locked for writing when this method is called (see also section 8.8 (page 153)). Use `after-resize` to modify the editor, if necessary.

Note that a snip calls `resized`, not this method, to notify the pasteboard that the snip resized itself.

- (send *a-pasteboard* `on-resize` *snip w h*) ⇒ void
   snip : `snip%` object
   w : non-negative real number
   h : non-negative real number

on-select

This method is called before a snip in the pasteboard is selected or deselected, after `can-select?` is called to verify that the selection is allowed. The `after-select` method is guaranteed to be called after the selection

has completed. This method is not called when a selected snip is to be deleted (and thus de-selected indirectly); see also `on-delete` .

The editor is internally locked for writing when this method is called (see also section 8.8 (page 153)). Use `after-select` to modify the editor, if necessary.

- (send *a-pasteboard* `on-select` *snip on?*) ⇒ void
    snip : `snip%` object
    on? : boolean

    If *on?* is `#t`, then *snip* will be selected, otherwise *snip* will be deselected.

raise

Moves a snip one level shallower (i.e., in front of one more other snip) in the pasteboard's snip order. See section 8.1 (page 148) for information about snip order in pasteboards.

See also `lower`, `set-before`, and `set-after`.

- (send *a-pasteboard* `raise` *snip*) ⇒ void
    snip : `snip%` object

remove

Removes the specified snip from the editor in a non-undoable manner (so the snip is completely free of the pasteboard can can be used in other editors).

See also `delete`.

- (send *a-pasteboard* `remove` *snip*) ⇒ void
    snip : `snip%` object

remove-selected

Deselects a snip without deselecting any other snips.

The selection in a pasteboard can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-select` to monitor selection changes.

- (send *a-pasteboard* `remove-selected` *snip*) ⇒ void
    snip : `snip%` object

    Deselects *snip* (if it is currently selected).

resize

Attempts to resize a given snip. If the snip allows resizing, `#t` is returned, otherwise `#f` is retured. Using this method instead of calling the snip's `resize` method directly will make the resize undo-able.

- (send *a-pasteboard* `resize` *snip w h*) ⇒ boolean
    snip : `snip%` object
    w : non-negative real number
    h : non-negative real number

set-after

Changes the depth of a snip. See section 8.1 (page 148) for information about snip order in pasteboards.

See also `raise`, `lower`, and `set-before`.

- (send *a-pasteboard* `set-after` *snip after*) ⇒ void
    snip : `snip%` object
    after : `snip%` object or `#f`

  Changes the depth of *snip* moving it just behind *after*. If *after* is `#f`, *snip* is moved to the back.

set-before

Changes the depth of a snip. See section 8.1 (page 148) for information about snip order in pasteboards.

See also `raise`, `lower`, and `set-after`.

- (send *a-pasteboard* `set-before` *snip before*) ⇒ void
    snip : `snip%` object
    before : `snip%` object or `#f`

  Changes the depth of *snip* moving it just in front of *before*. If *before* is `#f`, *snip* is moved to the front.

set-dragable

Sets whether snips in the editor can be interactively dragged by event handling in `on-default-event`: a true value allows dragging, `#f` disallows dragging. See also `get-dragable`.

- (send *a-pasteboard* `set-dragable` *allow-drag?*) ⇒ void
    allow-drag? : boolean

set-scroll-step

Sets the editor location offset for each vertical scroll position. See also `get-scroll-step`.

- (send *a-pasteboard* `set-scroll-step` *stepsize*) ⇒ void
    stepsize : non-negative real number

set-selected

Selects a specified snip (deselecting all others).

The selection in a pasteboard can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-select` to monitor selection changes.

- (send *a-pasteboard* `set-selected` *snip*) ⇒ void
    snip : `snip%` object

`set-selection-visible`

Sets whether selection dots are drawn around the edge of selected snips in the pasteboard. See also `get-selection-visible`.

- (send *a-pasteboard* `set-selection-visible` *visible?*) ⇒ void
    visible? : boolean

9.23   `snip%`

A direct instance of `snip%` is uninteresting. Useful snips are defined by instantiating derived subclasses, but this class defines the basic fucntionality.

In deriving a new snip class, these methods must be overridden to create a useful snip:

- `get-extent`

- `draw`

- `resize` if the snip can be resized by the user

- `partial-offset` if the snip can contain more than one item

- `split` if the snip can contain more than one item

- `size-cache-invalid` if the snip caches the result to `get-extent`

- `get-text` (not required)

- `find-scroll-step`, `get-num-scroll-steps`, and `get-scroll-step-offset` if the snip can contain more than one scroll position

If a snip can contain more than one item, then the snip's count must be maintained as well.

To define a class of snips that can be saved or cut-and-pasted:

- Create an instance of `snip-class%`, implementing the `read` method.

- For each instance of the snip class, set the snip's classs object with `set-snipclass`.

- Override the `copy` method.

- Override the `write` method.

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

    Creates a plain snip of length 1.

`adjust-cursor`

Called to determine the cursor image used when the cursor is moved over the snip in an editor. If `#f` is returned, a default cursor is selected by the editor. (See `adjust-cursor` in `editor<%>` for more information.)

- (send *a-snip* adjust-cursor *dc x y editorx editory event*) ⇒ cursor% object or #f
    dc : dc<%> object
    x : real number
    y : real number
    editorx : real number
    editory : real number
    event : mouse-event% object

    Returns #f.

blink-caret

Tells the snip to blink the selection caret. This method is called periodically when the snips's editor's display has the keyboard focus, and the snip has the editor-local focus.

The drawing context and snip's position in drawing context coordinates are provided.

- (send *a-snip* blink-caret *dc x y*) ⇒ void
    dc : dc<%> object
    x : real number
    y : real number

can-do-edit-operation?

See `can-do-edit-operation?`.

Called when the snip's editor's method is called, *recusive?* is not #f, and this snip owns the caret.

- (send *a-snip* can-do-edit-operation? *op recursive?*) ⇒ bool
    op : symbol   in   '(undo redo clear cut copy paste kill select-all
        insert-text-box insert-pasteboard-box insert-image)
    recursive? = #t : boolean

    See `can-do-edit-operation?` for information about the arguments.

copy

Creates and returns a copy of this snip. The `copy` method is responsible for copying this snip's style (as returned by `get-style`) to the new snip.

- (send *a-snip* copy) ⇒ snip% object

do-edit-operation

See `do-edit-operation`.

Called when the snip's editor's method is called, *recusive?* is not #f, and this snip owns the caret.

- (send *a-snip* do-edit-operation *op recursive?   time*) ⇒ void
    op : symbol   in   '(undo redo clear cut copy paste kill select-all
        insert-text-box insert-pasteboard-box insert-image)
    recursive? = #t : boolean
    time = 0 : exact integer

See `do-edit-operation` in `editor<%>` for information about the arguments.

draw

Called (by an editor) to draw the snip.

Before this method is called, the correct font, text color, and pen color will have been set in the drawing context for this snip already. (This is *not* true for `get-extent` or `partial-offset`.) The `draw` method must not make any other assumptions about the state of the drawing context, except that the clipping region is already set to something appropriate. Before `draw` returns, it must restore any drawing context settings that it changes.

See also `on-paint` in `editor<%>`.

The snip's editor is usually internally locked for writing and reflowing when this method is called (see also section 8.8 (page 153)).

- (send *a-snip* draw *dc x y left top right bottom dx dy draw-caret*) ⇒ void
    dc : `dc<%>` object
    x : real number
    y : real number
    left : real number
    top : real number
    right : real number
    bottom : real number
    dx : real number
    dy : real number
    draw-caret : symbol in '(no-caret show-inactive-caret show-caret)

    Draws the snip into the given drawing context with the snip's top left corner at location (x, y) in DC coordinates.

    The arguments *left*, *top*, *right*, and *bottom* define a clipping region (in DC coordinates) that the snip can use to optimize drawing, but it can also ignore these arguments.

    The *dx* and *dy* argument provide numbers that can be subtracted from *x* and *y* to obtain the snip's position in editor coordinates (as opposed to DC coordinates, which are used for drawing).

    See section 8.5 (page 152) for information about *draw-caret*.

find-scroll-step

If a snip contains more than one vertical scroll step (see `get-num-scroll-steps`) then this method is caled to find a scroll step offset for a given y-offset into the snip.

- (send *a-snip* find-scroll-step *y*) ⇒ exact non-negative integer
    y : real number

get-admin

Returns the administrator for this snip. (The administrator can be `#f` even if the snip is owned but not visible in the editor.)

- (send *a-snip* get-admin) ⇒ `snip-admin%` object or `#f`

`get-count`

Returns the number of items within the snip.

- - (`send` *a-snip* `get-count`) ⇒ exact integer in [0, 100000]

`get-extent`

Calculates the snip's width, height, descent (amount of height which is drawn below the baseline), space (amount of height which is "filler" space at the top), and horizontal spaces (amount of width which is "filler" space at the left and right).

A drawing context is provided for the purpose of finding font sizes, but no drawing should occur. The `get-extent` and `partial-offset` methods must not make any assumptions about the state of the drawing context, except that it is scaled properly. In particular, the font for the snip's style is not automatically set in the drawing context before the method is called.[2] If `get-extent` or `partial-offset` changes the drawing context's setting, it must restore them before returning. However, the methods should not need to change the drawing context; only font settings can affect measurement results from a device context, and `get-text-extent` in `dc<%>` accepts a `font%` argument for sizing that overrides that device context's current font.

The snip's left and top locations are provided in editor corrdinates. In a text editor, the y-coordinate is the *line's* top location; the snip's actual top location is potentially undetermined until its height is known.

This method is called by the snip's administrator; it should not be called directly by others. To get the extent of a snip, use `get-snip-location` in `editor<%>` .

The snip's editor is usually internally locked for writing and reflowing when this method is called (see also section 8.8 (page 153)).

- - (`send` *a-snip* `get-extent` *dc x y w h descent space lspace rspace*) ⇒ void
    dc : `dc<%>` object
    x : real number
    y : real number
    w = `#f` : boxed non-negative real number or `#f`
    h = `#f` : boxed non-negative real number or `#f`
    descent = `#f` : boxed non-negative real number or `#f`
    space = `#f` : boxed non-negative real number or `#f`
    lspace = `#f` : boxed non-negative real number or `#f`
    rspace = `#f` : boxed non-negative real number or `#f`

    Fills in all boxes with 0.0.

`get-flags`

Returns flags defining the behavior of the snip. It is a bitwise combination of these flags:

- `'is-text` — this is a text snip derived from `string-snip%`; do not set this flag

- `'can-append` — this snip can be merged with another snip of the same type

- `'invisible` — the user doesn't "see" this snip; e.g.: a carriage return

---

[2]Many snips cache their size information, so automatically setting the font would be wasteful.

- • 'hard-newline — a newline must follow the snip

- • 'newline — a newline currently follows the snip; only an owning editor should set this flag

- • 'handles-events — this snip can handle keyboard and mouse events

- • 'width-depends-on-x — this snip's display width depends on the snip's x-location within the editor; e.g.: tab

- • 'height-depends-on-y — this snip's display height depends on the snip's y-location within the editor

- • 'width-depends-on-y — this snip's display width depends on the snip's y-location within the editor

- • 'height-depends-on-x — this snip's display height depends on the snip's x-location within the editor

- • 'uses-editor-path — this snip uses its editor's pathname and should be notified when the name changes; notification is given as a redundant call to `set-admin`

Additional private flags are not listed here.

- (send *a-snip* get-flags) ⇒ list of symbols

get-num-scroll-steps

Returns the number of horizontal scroll steps within the snip. For most snips, this is 1. Embedded editor snips use this method so that scrolling in the owning editor will step through the lines in the embedded editor.

- (send *a-snip* get-num-scroll-steps) ⇒ exact non-negative integer

get-scroll-step-offset

If a snip contains more than one vertical scroll step (see `get-num-scroll-steps`) then this method is called to find the y-offset into the snip for a given scroll offset.

- (send *a-snip* get-scroll-step-offset *offset*) ⇒ non-negative real number
  offset : exact non-negative integer

get-snipclass

Returns the snip's class, used for file saving and cut-and-paste.

- (send *a-snip* get-snipclass) ⇒ snip-class% object

get-style

Returns the snip's style. See also `set-style`.

- (send *a-snip* get-style) ⇒ style<%> object

`get-text`

Gets the text representation for this snip.

> - (send *a-snip* `get-text` *offset num flattened?*) ⇒ string
>   *offset* : exact non-negative integer
>   *num* : exact non-negative integer
>   *flattened?* = `#f` : boolean

Returns the text for this snip starting with the item position *offset* within the snip, and continuing for a total length of *num* items. If *offset* is greater than the snip's count, then `""` is returned. If *num* is greater than the snip's count minus the offset, then text from the offset to the end of the snip is returned.

If *flattened?* is not `#f`, then flattened text is returned. See section 8.4 (page 152) for a discussion of flattened vs. non-flattened text.

`is-owned?`

Returns `#t` if this snip has an owner, `#f` otherwise. Note that a snip may be owned by an editor if it was inserted and then deleted from the editor, if it's still in the editor's undo history.

> - (send *a-snip* `is-owned?`) ⇒ boolean

`match?`

Return `#t` if this snip "matches" an input snip or `#f` otherwise.

> - (send *a-snip* `match?` *snip*) ⇒ boolean
>   *snip* : `snip%` object

Returns `#t` if the *snip* and this snip are from the same class and have the same length.

`merge-with`

Merges this snip with the given snip, returning `#f` if the snips cannot be merged or a new merged snip otherwise. This method will only be invoked if both snips are from the same class and both have the `'can-append` flag.

If the returned snip does not have the expected count, its count is forcibly modified. If the returned snip is already owned by a another administrator, a surrogate snip is created.

The snip's editor is usually internally locked for reading when this method is called (see also section 8.8 (page 153)).

> - (send *a-snip* `merge-with` *pred*) ⇒ `snip%` object or `#f`
>   *pred* : `snip%` object

Returns `#f`.

`next`

Returns the next snip in the editor owning this snip, or `#f` if this is the last snip.

In a text editor, the next snip is the snip at the text position following this snip's (last) text position. In a pasteboard, the next snip is the one immediately behind this snip. (See section 8.1 (page 148) for information about snip order in pasteboards.)

- (send *a-snip* next) ⇒ snip% object or #f

on-char

Called to handle keyboard events when this snip has the keyboard focus and can handle events. The drawing context is provided, as well as the snip's location in display coordinates (the event uses display coordinates), and the snip's location in editor coordinates.

See also 'handles-events in get-flags.

- (send *a-snip* on-char *dc* *x* *y* *editorx* *editory* *event*) ⇒ void
    dc : dc<%> object
    x : real number
    y : real number
    editorx : real number
    editory : real number
    event : key-event% object

  The *x* and *y* arguments are the snip's location in display coordiantes. The *editorx* and *editory* arguments are the snip's location in editor coordinates. To get *event*'s x location in snip coordinates, subtract *x* from (send *event* get-x).

on-event

Called to handle mouse events on the snip when this snip can handle events and when the snip has the keyboard focus. See on-char for information about the arguments. See also 'handles-events in get-flags.

- (send *a-snip* on-event *dc* *x* *y* *editorx* *editory* *event*) ⇒ void
    dc : dc<%> object
    x : real number
    y : real number
    editorx : real number
    editory : real number
    event : mouse-event% object

  The *x* and *y* arguments are the snip's location in display coordiantes. The *editorx* and *editory* arguments are the snip's location in editor coordinates. To get *event*'s x location in snip coordinates, subtract *x* from (send *event* get-x).

own-caret

Notifies the snip that it is or is not allowed to display the caret (indicating oversnip of keyboard focus) in some display. This method is *not* caled to request that the caret is actually shown or hidden; the draw method is called for all display requests.

- (send *a-snip* own-caret *own-it?*) ⇒ void
    own-it? : boolean

  The *own-it?* argument is #t if the snip owns the keyboard focus or #f otherwise.

239

`partial-offset`

Calculates a partial width for the snip, starting from the first snip item and continuing for a given number of items. The drawing context and snip's position in editor coordinates are provided. See also `get-extent`.

The snip's editor is usually internally locked for writing and reflowing when this method is called (see also section 8.8 (page 153)).

- (send *a-snip* `partial-offset` *dc x y len*) $\Rightarrow$ real number
  dc : `dc<%>` object
  x : real number
  y : real number
  len : exact non-negative integer

  Calculates a partial width for the snip, starting from the first snip item and continuing for *len* items.

`previous`

Returns the previous snip in the editor owning this snip, or `#f` if this is the first snip.

- (send *a-snip* `previous`) $\Rightarrow$ `snip%` object or `#f`

`release-from-owner`

Asks the snip to try to release itself from its owner. If the snip is not owned or the release is successful, then `#t` is returned. Otherwise, `#f` is returned and the snip remains owned. See also `is-owned?`.

Use this method for moving a snip from one editor to another. This method notifies the snip's owning editor that someone else really wants control of the snip. It is not necessary to use this method for "cleaning up" a snip when it is deleted from an editor.

- (send *a-snip* `release-from-owner`) $\Rightarrow$ boolean

  Requests a low-level release from the snip's owning administrator.

`resize`

Resizes the snip. The snip can refuse to be resized by returning `#f`. Otherwise, the snip will resize (it must call its administrator's `resized` method) and return `#t`.

See also `on-interactive-resize` in `pasteboard%`.

- (send *a-snip* `resize` *w h*) $\Rightarrow$ boolean
  w : non-negative real number
  h : non-negative real number

  Returns `#f`.

`set-admin`

Sets the snip's administrator. Only an administrator should call this method.

The default method sets the internal state of a snip to record its administrator. It will not modify this state if the snip is already owned by an administrator and the administrator has not blessed the transition. If

the administrator state of a snip is not modified as expected during a sensitive call to this method by an instance of `text%` or `pasteboard%`, the internal state may be forcibly modified (if the new administrator was `#f`) or a surrogate snip may be created (if the snip was expected to receive a new administrator).

The snip's (new) editor is usually internally locked for reading when this method is called (see also section 8.8 (page 153)).

- (send *a-snip* `set-admin` *admin*) ⇒ void
    admin : `snip-admin%` object or `#f`

set-count

Sets the number of items within the snip.

The snip's count may be changed by the system (in extreme cases to maintain consistency) without calling this method.

- (send *a-snip* `set-count` *c*) ⇒ void
    c : exact integer in [1, 100000]

    Sets the number of items in the snip, and notifies the snip's administrator that its size has changed.

set-flags

Sets the snip's flags. See `get-flags`.

- (send *a-snip* `set-flags` *flags*) ⇒ void
    flags : list of symbols

    Sets the snip flags and notifies the snip's editor that its flags have changed.

set-snipclass

Sets the snip's class, used for file saving and cut-and-paste.

- (send *a-snip* `set-snipclass` *class*) ⇒ void
    class : `snip-class%` object

set-style

Sets the snip's style if it is not owned by any editor. See also `get-style` and `is-owned?`.

The snip's style may be changed by the system without calling this method.

- (send *a-snip* `set-style` *style*) ⇒ void
    style : `style<%>` object

size-cache-invalid

Called to notify the snip that it may need to recalculate its display arguments (width, height, etc.) when it is next asked, because the style or location of the snip has changed.

The snip's (new) editor is usually internally locked for reflowing when this method is called (see also section 8.8 (page 153)).

- (send *a-snip* `size-cache-invalid`) ⇒ void

split

Splits the snip into two snips. This is called when a snip has more than one item and something is inserted between two items.

The arguments are a position integer and two boxes. The position integer specifies how many items should be given to the new first snip; the rest go to the new second snip. The two boxes must be filled with two new snips. (The old snip is no longer used, so it can be recycled as a new snip.)

If the returned snips do not have the expected counts, their counts are forcibly modified. If either returned snip is already owned by a another administrator, a surrogate snip is created.

The snip's editor is usually internally locked for reading when this method is called (see also section 8.8 (page 153)).

- (send *a-snip* `split` *position first second*) ⇒ void
   position : exact non-negative integer
   first : boxed `snip%` object
   second : boxed `snip%` object

write

Writes the snip to the given stream. (Snip reading is handled by the snip class.) Style information about the snip (i.e., the content of `get-style`) will be saved and restored automatically.

- (send *a-snip* `write` *f*) ⇒ void
   f : `editor-stream-out%` object

9.24 `snip-admin%`

See section 8.1.1 (page 148) for information about the role of administrators. The `snip-admin%` class is never instantiated directly. It is not even instantiated through derived classes by most programmers; each `text%` or `pasteboard%` object creates its own administrator. However, it may be useful to derive a new instance of this class to display snips in a new context. Also, it may be useful to call the methods of an existing administrator from an owned snip.

To create a new `snip-admin%` class, all methods described here must be overridden. They are all invoked by the administrator's snip.

Because a `snip-admin%` object typically owns more than one snip, many methods require a `snip%` object as a argument.

- (`make-object` `snip-admin%`) ⇒ `snip-admin%` object

   Creates a (useless) editor administrator.

`get-dc`

Gets a drawing context suitable for determining display size information. If the snip is not displayed, `#f` is returned.

- (send *a-snip-admin* `get-dc`) ⇒ `dc<%>` object or `#f`

`get-editor`

Returns the editor that this administrator reports to (directly or indirectly)

- (send *a-snip-admin* `get-editor`) ⇒ `text%` or `pasteboard%` object

`get-view`

Gets the position and size of the visible region of a snip in snip coordinates. The result is undefined if the given snip is not managed by this administrator.

If no snip is specified, then the position and size of the snip's editor are returned, instead, in editor coordinates.

See also `get-view` in `editor-admin%`.

- (send *a-snip-admin* `get-view` x y w h *snip*) ⇒ void
    x : boxed real number or `#f`
    y : boxed real number or `#f`
    w : boxed non-negative real number or `#f`
    h : boxed non-negative real number or `#f`
    snip = `#f` : `snip%` object or `#f`

If *snip* is not `#f`, the current visible region of the snip is installed in the boxes x, y, w, and h. The x and y values are relative to the snip's top-left corner. The w and h values may be larger than the snip itself.

If *snip* is `#f`, the total visible region of the snip's top-level display is returned in editor coordinates. Using `#f` for *snip* is anaologous to using `#t` for *full?* in `get-view` in `editor-admin%`.

`get-view-size`

Gets the visible size of the administrator's display region.

If the display is an editor canvas, see also `reflow-container`.

- (send *a-snip-admin* `get-view-size` h w) ⇒ void
    h : boxed non-negative real number or `#f`
    w : boxed non-negative real number or `#f`

`needs-update`

Called by the snip to request that the snip's display needs to be updated. The administrator determines when to actually update the snip; the snip's **draw**method is eventually called.

No update occurs if the given snip is not managed by this administrator.

- (send *a-snip-admin* `needs-update` *snip localx localy w h*) ⇒ void
    snip : `snip%` object
    localx : real number
    localy : real number
    w : non-negative real number
    h : non-negative real number

The *localx*, *localy*, *w*, and *h* arguments specify a region of the snip to be refreshed (in snip coordinates).

popup-menu

Opens a popup menu in the display for this snip's editor. The result is `#t` if the popup succeeds, `#f` otherwise (independent of whether the user selects an item in the popup menu).

While the menu is popped up, its target is set to the top-level editor in the display for this snip's editor. See `get-popup-target` for more information.

- (send *a-snip-admin* `popup-menu` *menu snip x y*) ⇒ bool
    menu : `popup-menu%` object
    snip : `snip%` object
    x : real number
    y : real number

The menu is displayed at *x* and *y* in *snip* coordinates.

recounted

Called by a snip to notify the administrator that the specified snip has changed its count. The snip generally needs to be updated after changing its count, but the snip decides whether the update should occur immediately.

The method call is ignored if the given snip is not managed by this administrator.

- (send *a-snip-admin* `recounted` *snip refresh?*) ⇒ void
    snip : `snip%` object
    refresh? : boolean

If *refresh?* is not `#f`, then the snip is requesting to be updated immediately. Otherwise, `needs-update` must eventually be called as well.

release-snip

Requests that the specified snip be released. If this administrator is not the snip's owner or if the snip cannot be released, then `#f` is returned. Otherwise, `#t` is returned and the snip is no longer owned.

See also `release-snip` in `editor<%>` .

The result is `#f` if the given snip is not managed by this administrator.

- (send *a-snip-admin* `release-snip` *snip*) ⇒ boolean
    snip : `snip%` object

`resized`

Called by a snip to notify the administrator that the specified snip has changed its display size (without being polled by `get-extent`). The snip generally needs to be updated after a resize, but the snip decides whether the update should occur immediately.

The method call is ignored if the given snip is not managed by this administrator.

- (send *a-snip-admin* `resized` *snip refresh?*) ⇒ void
    snip : `snip%` object
    refresh? : boolean

  If *refresh?* is not `#f`, then the snip is requesting to be updated immediately. Otherwise, `needs-update` must eventually be called as well.

`scroll-to`

Called by the snip to request scrolling so that the given region is visible. The snip generally needs to be updated after a scroll, but the snip decides whether the update should occur immediately.

The result is `#t` if the editor is scrolled, `#f` oitherwise.

The method call is ignored (and the result is `#f`) if the given snip is not managed by this administrator.

- (send *a-snip-admin* `scroll-to` *snip localx localy w h refresh? bias*) ⇒ boolean
    snip : `snip%` object
    localx : real number
    localy : real number
    w : non-negative real number
    h : non-negative real number
    refresh? : boolean
    bias = `'none` : symbol in `'(start end none)`

  The *localx*, *localy*, *w*, and *h* arguments specify a region of the snip to be made visible by the scroll (in snip coordinates).

  If *refresh?* is not `#f`, then the editor is requesting to be updated immediately.

  The *bias* argument is one of:

    − `'start` — if the range doesn't fit in the visible area, show the top-left region
    − `'none` — no special scrolling instructions
    − `'end` — if the range doesn't fit in the visible area, show the bottom-right region

`set-caret-owner`

Requests that the keyboard focus is assigned to the specified snip. If the request is granted, the `own-caret` method of the snip is called.

The method call is ignored if the given snip is not managed by this administrator.

- (send *a-snip-admin* `set-caret-owner` *snip domain*) ⇒ void
    snip : `snip%` object
    domain : symbol in `'(immediate display global)`

  See `set-caret-owner` for imformation about the possible values of *domain*.

update-cursor

Queues an update for the cursor in the display for this snip's editor. The actual cursor used will be determined by calling the snip's `adjust-cursor`method as appropriate.

- (send *a-snip-admin* update-cursor) ⇒ void

9.25   `snip-class%`

Useful snip classes are defined by instantiating derived subclasses of `snip-class%`. A class derived from `snip-class%` serves as a kind of "meta-class" for snips; each snip is associated with an instance of `snip-class%` as its snip class.

In deriving a new `snip-class%` class, override the `read` method. Then, for each instance of the derived class (where each instance corresponds to a single snip class):

- Set the classname using `set-classname`.

- Set the version using `set-version`.

- Install the class into the list returned by `get-the-snip-class-list` using the `add` method. Note that if the same name is inserted into the same class list mulitple times, all but the first insertion is ignored.

See also section 8.2.1 (page 150).

- (make-object snip-class%) ⇒ snip-class% object

    Creates a useless snip class.

get-classname

Return's the class's name, a string uniquely designating this snip class. For example, the standard text snip classname is "wxtext". Names beginning with "wx" are reserved.

- (send *a-snip-class* get-classname) ⇒ string

get-version

Returns the version of this snip class. When attempting to load a file containing a snip with the same class name but a different version, the user is warned.

- (send *a-snip-class* get-version) ⇒ exact integer

read

Reads a snip from a given stream, returning a newly created snip as the result or `#f` if there is an error.

- (send *a-snip-class* read *f*) ⇒ snip% object or #f
    f : editor-stream-in% object

read-header

Called to read header information that may be useful for every snip read in this class. This method is only called once per editor read session, and only if the stream contains header information for this class.

The return value is `#f` if a read error occurs or anything else otherwise.

See also `write-header`.

- (send *a-snip-class* `read-header` *f*) ⇒ boolean
    f : `editor-stream-in%` object

reading-version

Returns the version number specified for this snip class for snips currently being read from the given stream.

- (send *a-snip-class* `reading-version` *class*) ⇒ exact integer
    class : `editor-stream-in%` object

set-classname

Sets the class's name. See also `get-classname`.

- (send *a-snip-class* `set-classname` *name*) ⇒ void
    name : string

set-version

Sets the version of this class. See `get-version`.

- (send *a-snip-class* `set-version` *v*) ⇒ void
    v : exact integer

write-header

Called to write header information that may be useful for every snip written for this class. This method is only called once per editor write session, and only if the editor contains snips in this class.

When reading the snips back in, `read-header` will only be called if `write-header` writes some data to the stream.

The return value is `#f` if a write error occurs or anything else otherwise.

- (send *a-snip-class* `write-header` *stream*) ⇒ boolean
    stream : `editor-stream-out%` object

9.26   `snip-class-list<%>`

Each eventspace has its own instance of `snip-class-list<%>`, obtained with `(get-the-snip-class-list)`. New instances cannot be created directly. Each instance keeps a list of snip classes. This list is needed for loading snips from a file. See also section 8.2.1 (page 150).

`add`

Adds a snip class to the list. If a class with the same name already exists in the list, this one will not be added.

- (`send` *a-snip-class-list* `add` *snipclass*) ⇒ void
    snipclass : `snip-class%` object

`find`

Finds a snip class from the list with the given name, returning `#f` if none is found.

- (`send` *a-snip-class-list* `find` *name*) ⇒ `snip-class%` object or `#f`
    name : string

`find-position`

Returns an index into the list for the specified class.

- (`send` *a-snip-class-list* `find-position` *class*) ⇒ exact non-negative integer
    class : `snip-class%` object

`nth`

Returns the *n*th class in the list, or `#f` if the list has *n* classes or less.

- (`send` *a-snip-class-list* `nth` *n*) ⇒ `snip-class%` object or `#f`
    n : exact non-negative integer

`number`

Returns the number of snip classes in the list.

- (`send` *a-snip-class-list* `number`) ⇒ exact non-negative integer

9.27 `string-snip%`

Superclass: `snip%`

An instance of `string-snip%` is created automatically when text is inserted into a text editor. See also `on-new-string-snip` in `text%`.

- (`make-object` `string-snip%` *allocsize*) ⇒ `string-snip%` object
    allocsize = 0 : exact non-negative integer

  Creates an empty string snip. The *allocsize* argument is a hint about how much storage space for text should be initially allocated by the snip.

- (`make-object` `string-snip%` *s*) ⇒ `string-snip%` object
    s : string

  Creates a string snip with the given initial string.

`insert`

Inserts text into the snip. The system can insert text into a text snip without calling this method.

- (send *a-string-snip* `insert` *s len pos*) ⇒ void
  - *s* : string
  - *len* : exact non-negative integer
  - *pos* = 0 : exact non-negative integer

  Inserts *s* (with length *len*) into the snip at position *pos* within the snip.

`read`

Reads the snip's data from the given stream.

- (send *a-string-snip* `read` *len f*) ⇒ void
  - *len* : exact non-negative integer
  - *f* : `editor-stream-in%` object

  The *len* argument specifies the maximum length of the text to be read. (When a text snip is written to a file, the very first field is the length of the text contained in the snip.) This method is usually invoked by the text snip class's `read` method.

9.28   `style<%>`

A `style<%>` object encapsulates drawing information (font, color, alignment, etc.) in a hierarchical manner. A `style<%>` object always exists within the context of a `style-list%` object and is never created except by a `style-list%` object.

See also section 8.1.2 (page 149).

`get-alignment`

Returns the style's alignment: 'top, 'center, or 'bottom.

- (send *a-style* `get-alignment`) ⇒ symbol in '(top center bottom)

`get-background`

Returns the style's background color.

- (send *a-style* `get-background`) ⇒ `color%` object

`get-base-style`

Returns the style's base style. See section 8.1.2 (page 149) for more information. The return value is `#f` only for the basic style in the list.

- (send *a-style* `get-base-style`) ⇒ `style<%>` object or `#f`

get-delta

Returns the style's delta information if the style is not a join style. See section 8.1.2 (page 149) for more information.

- (send *a-style* get-delta *delta*) ⇒ void
    delta : style-delta% object

  Copies the style's delta into *delta*.

get-face

Returns the style's face name. See font%.

- (send *a-style* get-face) ⇒ string or #f

get-family

Returns the style's font family. See font%.

- (send *a-style* get-family) ⇒ symbol in '(default decorative roman script swiss modern symbol system)

get-font

Returns the style's font information.

- (send *a-style* get-font) ⇒ font% object

get-foreground

Returns the style's foreground color.

- (send *a-style* get-foreground) ⇒ color% object

get-name

Returns the style's name, or #f if it is unnamed. Style names are only set through the style's style-list% object.

- (send *a-style* get-name) ⇒ string or #f

get-shift-style

Returns the style's shift style if it is a join style. Otherwise, the root style is returned. See section 8.1.2 (page 149) for more information.

- (send *a-style* get-shift-style) ⇒ style<%> object

`get-size`

Returns the style's font size.

- (send *a-style* `get-size`) ⇒ exact integer in [0, 255]

`get-style`

Returns the style's font style. See `font%`.

- (send *a-style* `get-style`) ⇒ symbol in '(`normal italic slant`)

`get-text-descent`

Returns the descent of text using this style in a given DC.

- (send *a-style* `get-text-descent` *dc*) ⇒ non-negative real number
  dc : `dc<%>` object

`get-text-height`

Returns the height of text using this style in a given DC.

- (send *a-style* `get-text-height` *dc*) ⇒ non-negative real number
  dc : `dc<%>` object

`get-text-space`

Returns the vertical spacing for text using this style in a given DC.

- (send *a-style* `get-text-space` *dc*) ⇒ non-negative real number
  dc : `dc<%>` object

`get-text-width`

Returns the width of a space character using this style in a given DC.

- (send *a-style* `get-text-width` *dc*) ⇒ non-negative real number
  dc : `dc<%>` object

`get-transparent-text-backing`

Returns `#t` if text is drawn without erasing the text background or `#f` otherwise.

- (send *a-style* `get-transparent-text-backing`) ⇒ boolean

get-underlined

Returns #t if the style is underlined or #f otherwise.

- (send *a-style* get-underlined) ⇒ boolean

get-weight

Returns the style's font weight. See font%.

- (send *a-style* get-weight) ⇒ symbol in '(normal bold light)

is-join?

Returns #t if the style is a join style or #f otherwise. See section 8.1.2 (page 149) for more information.

- (send *a-style* is-join?) ⇒ boolean

set-base-style

Sets the style's base style and recomputes the style's font, etc. See section 8.1.2 (page 149) for more information.

- (send *a-style* set-base-style *base-style*) ⇒ void
    base-style : style<%> object

set-delta

Sets the style's delta (if it is not a join style) and recomputes the style's font, etc. See section 8.1.2 (page 149) for more information.

- (send *a-style* set-delta *delta*) ⇒ void
    delta : style-delta% object

  Copies *delta* into the style's delta.

set-shift-style

Sets the style's shift style (if it is a join style) and recomputes the style's font, etc. See section 8.1.2 (page 149) for more information.

- (send *a-style* set-shift-style *style*) ⇒ void
    style : style<%> object

switch-to

Sets the font, pen color, etc. of the given drawing context. If *oldstyle* is not #f, only differences between the given style and this one are applied to the drawing context.

- (send *a-style* `switch-to` *dc old-style*) ⇒ void
  dc : `dc<%>` object
  old-style : `style<%>` object or `#f`

9.29 `style-delta%`

A `style-delta%` object encapsulates a style change. The changes expressible by a delta include:

- changing the font family
- changing the font face
- changing the font size to a new value
- enlarging the font by an additive amount
- enlarging the font by a multiplicative amount, etc.
- changing the font style (normal, *italic*, or *slant*)
- toggling the font style
- changing the font to *italic* if it is currently *slant*, etc.
- changing the font weight, etc.
- changing the underline, etc.
- changing the vertical alignment, etc.
- changing the foreground color
- dimming or brightening the foreground color, etc.
- changing the background color, etc.
- changing text backing transparency

The `set-delta` method is convenient for most style delta settings; it takes a high-level delta specification and sets the internal delta information.

To take full advantage of a style delta, it is necessary to understand the internal on/off settings that can be manipulated through methods such as `set-weight-on`. For example, the font weight change is specified through the `weight-on` and `weight-off` internal settings. Roughly, `weight-on` turns on a weight setting when it is not present and `weight-off` turns off a weight setting when it is present. These two interact precisely in the following way:

- If both `weight-on` and `weight-off` are set to `'base`, then the font weight is not changed.
- If `weight-on` is not `'base`, then the weight is set to `weight-on`.
- If `weight-off` is not `'base`, then the weight will be set back to `'normal` when the base style has the weight `weight-off`.
- If both `weight-on` and `weight-off` are set to the same value, then the weight is toggled with respect to that value: if the base style has the weight `weight-on`, then weight is changed to `'normal`; if the base style has a different weight, it is changed to `weight-on`.
- If both `weight-on` and `weight-off` are set, but to different values, then the weight is changed to `weight-on` only when the base style has the weight `weight-off`.

Font styles, underlining, and alignment work in an analogous manner.

The possible values for `alignment-on` and `alignment-off` are:

- `'base`

- `'top`

- `'center`

- '`bottom`

The possible values for `style-on` and `style-off` are:

- '`base`

- '`normal`

- '`italic`

- '`slant`

The possible values for `underlined-on` and `underlined-off` are:

- `#f` (acts like '`base`)

- `#t`

The possible values for `trasnparent-text-backing-on` and `trasnparent-text-backing-off` are:

- `#f` (acts like '`base`)

- `#t`

The possible values for `weight-on` and `weight-off` are:

- '`base`

- '`normal`

- '`bold`

- '`light`

The family and face settings in a style delta are interdependent:

- When a delta's face is `#f` and its family is '`base`, then neither the face nor family are modified by the delta.

- When a delta's face is a string and its family is '`base`, then only face is modified by the delta.

- When a delta's family is not '`base`, then both the face and family are modified by the delta. If the delta's face is `#f`, then applying the delta sets a style's face to `#f`, so that the family setting prevails in choosing a font.

- (make-object style-delta% *change-command*) $\Rightarrow$ style-delta% object
    change-command = '`change-nothing` : symbol   in   '(change-nothing change-normal change-toggle-unde
                                    change-normal-color change-bold)

- (make-object style-delta% *change-command v*) $\Rightarrow$ style-delta% object
    change-command : symbol   in   '(change-family change-style change-toggle-style
                        change-weight change-toggle-weight change-alignment)
    v : symbol

- (make-object `style-delta%` *change-command* *v*) ⇒ `style-delta%` object
  change-command : symbol in '(`change-size change-bigger change-smaller`)
  v : exact integer in [0, 255]

- (make-object `style-delta%` *change-command* *v*) ⇒ `style-delta%` object
  change-command : symbol in '(`change-underline`)
  v : boolean

The initialization arguments are passed on to `set-delta`.

collapse

Tries to collapse into a single delta the changes that would be made by applying this delta after a given delta. If the return value is `#f`, then it is impossible to perform the collapse. Otherwise, the return value is `#t` and this delta will contain the collapsed change specification.

- (send *a-style-delta* `collapse` *delta*) ⇒ boolean
  delta : `style-delta%` object

copy

Copies the given style delta's settings into this one.

- (send *a-style-delta* `copy` *delta*) ⇒ void
  delta : `style-delta%` object

equal?

Returns `#t` if the given delta is equivalent to this one in all contexts or `#f` otherwise.

- (send *a-style-delta* `equal?` *delta*) ⇒ boolean
  delta : `style-delta%` object

get-alignment-off

See `style-delta%`.

- (send *a-style-delta* `get-alignment-off`) ⇒ symbol in '(`base top center bottom`)

get-alignment-on

See `style-delta%`.

- (send *a-style-delta* `get-alignment-on`) ⇒ symbol in '(`base top center bottom`)

get-background-add

Gets the object additive color shift for the background (applied after the multiplicative factor). Call this `add-color<%>` object's methods to change the style delta's additive backgound color shift.

- (send *a-style-delta* `get-background-add`) ⇒ `add-color<%>` object

get-background-mult

Gets the multiplicative color shift for the background (applied before the additive factor). Call this `mult-color<%>` object's methods to change the style delta's multiplicative background color shift.

- (send *a-style-delta* get-background-mult) ⇒ `mult-color<%>` object

get-face

Gets the delta's font face string. If this string is `#f` and the family is `'base` when the delta is applied to a style, the style's face and family are not changed. However, if the gace string is `#f` and the family is not `'base`, then the style's face is changed to `#f`.

See also `get-family`.

- (send *a-style-delta* get-face) ⇒ string or `#f`

get-family

Returns the delta's font family. The possible values are

- `'base` — no change to family

- `'default`

- `'decorative`

- `'roman`

- `'script`

- `'swiss`

- `'modern` (fixed width)

- `'symbol` (Greek letters)

- `'system` (used to draw control labels)

See also `get-face`.

- (send *a-style-delta* get-family) ⇒ symbol in `'(base default decorative roman script swiss modern symbol system)`

get-foreground-add

Gets the additive color shift for the foreground (applied after the multiplicative factor). Call this `add-color<%>` object's methods to change the style delta's additive foreground color shift.

- (send *a-style-delta* get-foreground-add) ⇒ `add-color<%>` object

`get-foreground-mult`

Gets the multiplicative color shift for the foreground (applied before the additive factor). Call this `mult-color<%>` object's methods to change the style delta's multiplicative foreground color shift.

- (send *a-style-delta* `get-foreground-mult`) ⇒ `mult-color<%>` object

`get-size-add`

Gets the additive font size shift (applied after the multiplicative factor).

- (send *a-style-delta* `get-size-add`) ⇒ exact integer in [0, 255]

`get-size-mult`

Gets the multiplicative font size shift (applied before the additive factor).

- (send *a-style-delta* `get-size-mult`) ⇒ real number

`get-style-off`

See `style-delta%`.

- (send *a-style-delta* `get-style-off`) ⇒ symbol in '(base normal italic slant)

`get-style-on`

See `style-delta%`.

- (send *a-style-delta* `get-style-on`) ⇒ symbol in '(base normal italic slant)

`get-transparent-text-backing-off`

See `style-delta%`.

- (send *a-style-delta* `get-transparent-text-backing-off`) ⇒ boolean

`get-transparent-text-backing-on`

See `style-delta%`.

- (send *a-style-delta* `get-transparent-text-backing-on`) ⇒ boolean

`get-underlined-off`

See `style-delta%`.

- (send *a-style-delta* `get-underlined-off`) ⇒ boolean

`get-underlined-on`

See `style-delta%`.

- (send *a-style-delta* `get-underlined-on`) ⇒ boolean

`get-weight-off`

See `style-delta%`.

- (send *a-style-delta* `get-weight-off`) ⇒ symbol in '(base normal bold light)

`get-weight-on`

See `style-delta%`.

- (send *a-style-delta* `get-weight-on`) ⇒ symbol in '(base normal bold light)

`set-alignment-off`

See `style-delta%`.

- (send *a-style-delta* `set-alignment-off` *v*) ⇒ void
    v : symbol in '(base top center bottom)

`set-alignment-on`

See `style-delta%`.

- (send *a-style-delta* `set-alignment-on` *v*) ⇒ void
    v : symbol in '(base top center bottom)

`set-delta`

Configures the delta with high-level specifications. The return value is the delta itself.

Except for `'change-nothing` and `'change-normal`, the command only changes part of the delta. Thus, applying `'change-bold` and then `'change-italic` sets the delta for both the style and weight change.

- (send *a-style-delta* `set-delta` *change-command*) ⇒ `style-delta%` object
    change-command = '`change-nothing` : symbol   in   '(change-nothing change-normal change-toggle-unde
                                        change-normal-color change-bold)
  The *change-command* argument specifies how the delta is changed; the possible values are:
    – `'change-nothing` — reset all changes
    – `'change-normal` — turn off all styles and resizings
    – `'change-toggle-underline` — underline regions that are currently not underlined, and vice-versa
    – `'change-normal-color` — change the foreground and background to black and white, respectively

- − 'change-italic — change the style of the font to *italic*
- − 'change-bold — change the weight of the font to **bold**

- (send *a-style-delta* set-delta *change-command param*) ⇒ style-delta% object
  change-command : symbol   in   '(change-family change-style change-toggle-style change-weight change-toggle-weight change-alignment)
  param : symbol

  The *change-command* argument specifies how the delta is changed; the possible values are:
  - − 'change-family — change the font family (*param* is a family; see font%); see also get-family
  - − 'change-style — change the style of the font (*param* is a style; see font%)
  - − 'change-toggle-style — toggle the style of the font (*param* is a style; see font%)
  - − 'change-weight — change the weight of the font (*param* is a weight; see font%)
  - − 'change-toggle-weight — toggle the weight of the font (*param* is a weight; see font%)
  - − 'change-alignment — change the alignment(*param* is an alignment; see style-delta%)

- (send *a-style-delta* set-delta *change-command param*) ⇒ style-delta% object
  change-command : symbol in '(change-size change-bigger change-smaller)
  param : exact integer in [0, 255]

  The *change-command* argument specifies how the delta is changed; the possible values are:
  - − 'change-size — change the size to an absolute value (*param* is a size)
  - − 'change-bigger — make the text larger (*param* is an additive amount)
  - − 'change-smaller — make the text smaller (*param* is an additive amount)

- (send *a-style-delta* set-delta *change-command underlined?*) ⇒ style-delta% object
  change-command : symbol in '(change-underline)
  underlined? : boolean

  There is only one possible value for *change-command*:
  - − 'change-underline — set the underline status to either underlined or plain

`set-delta-background`

Makes the delta encode a background color change to the absolute color given. The return value is the delta itself.

- (send *a-style-delta* set-delta-background *name*) ⇒ style-delta% object
  name : string

  The string is looked up in the-color-database. See color-database<%>.

- (send *a-style-delta* set-delta-background *color*) ⇒ style-delta% object
  color : color% object

  The *color* argument is copied into the delta's background color.

`set-delta-face`

Like set-face, but sets the family at the same time.

The return value is the delta itself.

- (send *a-style-delta* set-delta-face *name family*) ⇒ style-delta% object
  name : string
  family = default : symbol   in   '(base default decorative roman script swiss modern symbol system)

`set-delta-foreground`

Makes the delta encode a foreground color change to the absolute color given. The return value is the delta itself.

- (send *a-style-delta* `set-delta-foreground` *name*) ⇒ `style-delta%` object
    name : string

  The string is looked up in `the-color-database`. See `color-database<%>`.

- (send *a-style-delta* `set-delta-foreground` *color*) ⇒ `style-delta%` object
    color : `color%` object

  The *color* argument is copied into the delta's foreground color.

`set-face`

See `get-face`. See also `set-delta-face`.

- (send *a-style-delta* `set-face` *v*) ⇒ void
    v : string or `#f`

`set-family`

Sets the delta's font family. See `get-family`.

- (send *a-style-delta* `set-family` *v*) ⇒ void
    v : symbol in `'(base default decorative roman script swiss modern symbol system)`

`set-size-add`

Sets the additive font size shift (applied after the multiplicative factor).

- (send *a-style-delta* `set-size-add` *v*) ⇒ void
    v : exact integer in [0, 255]

`set-size-mult`

Sets the multiplicative font size shift (applied before the additive factor).

- (send *a-style-delta* `set-size-mult` *v*) ⇒ void
    v : real number

`set-style-off`

See `style-delta%`.

- (send *a-style-delta* `set-style-off` *v*) ⇒ void
    v : symbol in `'(base normal italic slant)`

`set-style-on`

See `style-delta%`.

- (send *a-style-delta* `set-style-on` *v*) ⇒ void
    v : symbol in '(base normal italic slant)

`set-transparent-text-backing-off`

See `style-delta%`.

- (send *a-style-delta* `set-transparent-text-backing-off` *v*) ⇒ void
    v : boolean

`set-transparent-text-backing-on`

See `style-delta%`.

- (send *a-style-delta* `set-transparent-text-backing-on` *v*) ⇒ void
    v : boolean

`set-underlined-off`

See `style-delta%`.

- (send *a-style-delta* `set-underlined-off` *v*) ⇒ void
    v : boolean

`set-underlined-on`

See `style-delta%`.

- (send *a-style-delta* `set-underlined-on` *v*) ⇒ void
    v : boolean

`set-weight-off`

See `style-delta%`.

- (send *a-style-delta* `set-weight-off` *v*) ⇒ void
    v : symbol in '(base normal bold light)

`set-weight-on`

See `style-delta%`.

- (send *a-style-delta* `set-weight-on` *v*) ⇒ void
    v : symbol in '(base normal bold light)

9.30   `style-list%`

A `style-list%` object contains a set of `style<%>` objects and maintains the hierarchical relationships between them. A `style<%>` object can only be created through the methods of a `style-list%` object. There is a global style list object, `the-style-list`, but any number of independent lists can be created for separate style hierarchies. Each editor creates its own private style list.

See section 8.1.2 (page 149) for more information.

- (make-object style-list%) ⇒ style-list% object

    The root style, named `"Basic"`, is automatically created.

`basic-style`

Returns the root style. Each style list has its own root style.

- (send *a-style-list* basic-style) ⇒ style<%> object

`convert`

Converts an external style to a style in this list.

- (send *a-style-list* convert *style*) ⇒ style<%> object
    style : style<%> object

    Converts *style*, which can be from another style list, to a style in this list. If *style* is already in this list, then *style* is returned. If *style* is named and a style by that name is already in this list, then the existing named style is returned. Otherwise, the style is converted by converting it's base style (and shift style if *style* is a join style) and then creating a new style in this list.

`find-named-style`

Finds a style by name. If no such style can be found, `#f` is returned.

- (send *a-style-list* find-named-style *name*) ⇒ style<%> object or #f
    name : string

`find-or-create-join-style`

Creates a new join style, or finds an appropriate existing one. The returned style is always unnamed. See section 8.1.2 (page 149) for more information.

- (send *a-style-list* find-or-create-join-style *base-style* *shift-style*) ⇒ style<%> object
    base-style : style<%> object
    shift-style : style<%> object

    The *base-style* argument must be a style within this style list.

`find-or-create-style`

Creates a new derived style, or finds an appropriate existing one. The returned style is always unnamed. See section 8.1.2 (page 149) for more information.

- (send *a-style-list* find-or-create-style *base-style delta*) ⇒ style<%> object
  base-style : style<%> object
  delta : style-delta% object

The *base-style* argument must be a style within this style list.

forget-notification

See `notify-on-change`.

- (send *a-style-list* forget-notification *key*) ⇒ void
  key : value

The *key* argument is the value returned by `notify-on-change`.

index-to-style

Returns the style associated with the given index, or `#f` for a bad index. See also `style-to-index`.

- (send *a-style-list* index-to-style *i*) ⇒ style<%> object or #f
  i : exact non-negative integer

new-named-style

Creates a new named style, unless the name is already being used.

- (send *a-style-list* new-named-style *name like-style*) ⇒ style<%> object
  name : string
  like-style : style<%> object

If *name* is already being used, then *like-style* is ignored and the old style associated to the name is returned. Otherwise, a new style is created for *name* with the same characteristics (i.e., the same base style and same style delta or shift style) as *like-style*.

The *like-style* style must be in this style list, otherwise the named style is derived from the basic style with an empty style delta.

notify-on-change

Attaches a callback to the style list. The callback is invoked whenever a style is modified.

Often, a change in one style will trigger a change in several other derived styles; to allow clients to handle all the changes in a batch, `#f` is passed in as the changing style after a set of styles has been processed.

The return value from `notify-on-change` is an opaque key to be used with `forget-notification`.

- (send *a-style-list* notify-on-change *f*) ⇒ value
  f : procedure of one argument: a style<%> object or #f

number

Returns the number of styles in the list.

- (send *a-style-list* number) ⇒ exact non-negative integer

`replace-named-style`

Like `new-named-style`, except that if the name is already mapped to a style, the existing mapping is replaced.

- (send *a-style-list* `replace-named-style` *name like-style*) ⇒ `style<%>` object
  name : string
  like-style : `style<%>` object

`style-to-index`

Returns the index for a particular style. The index for a style's base style (and shift style, if it is a join style) is guaranteed to be lower than the style's own index. (As a result, the root style's index is always `0`.) An style's index can change whenever a new style is added to the list, or the base style or shift style of another style is changed.

If the given style is not in this list, `#f` is returned.

- (send *a-style-list* `style-to-index` *style*) ⇒ exact non-negative integer or `#f`
  style : `style<%>` object

9.31 `tab-snip%`

Superclass: `string-snip%`

An instance of `tab-snip%` is created automatically when a tab is inserted into an editor.

- (make-object `tab-snip%`) ⇒ `tab-snip%` object

  Creates a snip for a single tab.

9.32 `text%`

Implements: `editor<%>`

A `text%` object is a standard text editor. A text editor is displayed on the screen through a `editor-canvas%` object or some other display.

- (make-object `text%` *line-spacing tabstops*) ⇒ `text%` object
  line-spacing = `1.0` : non-negative real number
  tabstops = `null` : list of real numbers

  The *line-spacing* argument sets the additional amount of space (in DC units) inserted between each line in the editor when the editor is displayed. This spacing is included in the reported height of each line.

  See `set-tabs` for information about *tabstops*.

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

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

`after-change-style`

Called after the style is changed for a given range (and after the display is refreshed; use `on-change-style` and `begin-edit-sequence` to avoid extra refreshes when `after-change-style` modifies the editor).

See also `can-change-style?` and `on-edit-sequence`.

No internals locks are set when this method is called.

> - (send *a-text* `after-change-style` *start len*) ⇒ void
>   *start* : exact non-negative integer
>   *len* : exact non-negative integer

`after-delete`

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

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

No internals locks are set when this method is called.

> - (send *a-text* `after-delete` *start end*) ⇒ void
>   *start* : exact non-negative integer
>   *end* : exact non-negative integer
>
> The *start* argument specifies the starting position of the deleted range. The *len* argument specifies number of deleted items (so *start* + *length* is the endig position of the deleted range).

`after-insert`

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

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

No internals locks are set when this method is called.

> - (send *a-text* `after-insert` *start len*) ⇒ void
>   *start* : exact non-negative integer
>   *len* : exact non-negative integer
>
> The *start* argument specifies the position of the insert. The *len* argument specifies the total length (in positions) of the inserted items.

`after-set-position`

Called after the start and end position have been moved (but not when the position is moved due to inserts or deletes).

See also `on-edit-sequence`.

> - (send *a-text* `after-set-position`) ⇒ void

`after-set-size-constraint`

Called after the editor's maximum or minimum height or width is changed (and after the display is refreshed; use `on-set-size-constraint` and `begin-edit-sequence` to avoid extra refreshes when `after-set-size-constraint` modifies the editor).

(This callback method is provided because setting an editor's maxmimum width may cause lines to be re-flowed with soft carriage returns.)

See also `can-set-size-constraint?` and `on-edit-sequence`.

- (send *a-text* `after-set-size-constraint`) ⇒ void

`call-clickback`

Simulates a user click that invokes a clickback, if the given range of positions is within a clickback's region. See also section 8.7 (page 153).

- (send *a-text* `call-clickback` *start end*) ⇒ void
    start : exact non-negative integer
    end : exact non-negative integer

`can-change-style?`

Called before the style is changed in a given range of the editor. If the return value is `#f`, then the style change will be aborted.

The editor is internally locked for writing during a call to this method (see also section 8.8 (page 153)). Use `after-change-style` to modify the editor, if necessary.

See also `on-change-style`, `after-change-style`, and `on-edit-sequence`.

- (send *a-text* `can-change-style?`   *start len*) ⇒ boolean
    start : exact non-negative integer
    len : exact non-negative integer

`can-delete?`

Called before a range is deleted from the editor. If the return value is `#f`, then the delete will be aborted.

The editor is internally locked for writing during a call to this method (see also section 8.8 (page 153)). Use `after-delete` to modify the editor, if necessary.

See also `on-delete`, `after-delete`, and `on-edit-sequence`.

- (send *a-text* `can-delete?`   *start len*) ⇒ boolean
    start : exact non-negative integer
    len : exact non-negative integer

   The *start* argument specifies the starting position of the range to delete. The *len* argument specifies number of items to delete (so *start* + *length* is the endig position of the range to delete).

`can-insert?`

Called before items are inserted into the editor. If the return value is #f, then the insert will be aborted.

The editor is internally locked for writing during a call to this method (see also section 8.8 (page 153)). Use `after-insert` to modify the editor, if necessary.

See also `on-insert`, `after-insert`, and `on-edit-sequence`.

- (send *a-text* `can-insert?` *start len*) ⇒ boolean
    start : exact non-negative integer
    len : exact non-negative integer

  The *start* argument specifies the position of the potential insert. The *len* argument specifies the total length (in positions) of the items to be inserted.

`can-set-size-constraint?`

Called before the editor's maximum or minimum height or width is changed. If the return value is #f, then the change will be aborted.

(This callback method is provided because setting an editor's maxmimum width may cause lines to be re-flowed with soft carriage returns.)

See also `on-set-size-constraint`, `after-set-size-constraint`, and `on-edit-sequence`.

- (send *a-text* `can-set-size-constraint?`) ⇒ boolean

`caret-hidden?`

Returns #t if the caret is hidden for this editor or #f otherwise.

- (send *a-text* `caret-hidden?`) ⇒ boolean
    See also `hide-caret`.

`change-style`

Changes the style for items in the editor.

The style within an editor can be changed by the system (in response to other method calls), and such changes do not go through this method; use `on-change-style` in `text%` to monitor style changes.

- (send *a-text* `change-style` *delta start end*) ⇒ void
    delta : `style-delta%` object
    start : exact non-negative integer or 'start
    end = 'end : exact non-negative integer or 'end

  Changes the style for a region in the editor by applying a style delta. If *start* is 'start and *end* is 'end, then the currently selected items are changed. Otherwise, if *end* is 'end, then the style is changed from *start* until the end of the selection.

- (send *a-text* `change-style` *style start end*) ⇒ void
    style : `style<%>` object

>   *start* = 'start : exact non-negative integer or 'start
>   *end* = 'end : exact non-negative integer or 'end

Changes the style for a region in the editor to a specific style. If *start* is 'start and *end* is 'end, then the currently selected items are changed. Otherwise, if *end* is 'end, then the style is changed from *start* until the end of the selection.

- (send *a-text* change-style *delta*) ⇒ void
     delta : style-delta% object

Changes the style of the selected items by applying a style delta.

To change a large collection of snips from one style to another style, consider providing a style<%> instance rather than a style-delta% instance. Otherwise, change-style must convert the style-delta% instance to the style<%> instance for every snip; this conversion consumes both time and (temporary) memory.

- (send *a-text* change-style *style*) ⇒ void
     style : style<%> object

Changes the style of the selected items to a specific style. The editor's style list must contain *style*, otherwise the style is not changed. See also convert.

copy

Copies items into the clipboard.

The system may execute a copy (in response to other method calls) without calling this method. To extend or re-implement copying, override the do-copy in text% or do-copy in pasteboard% method of an editor.

- (send *a-text* copy *extend?* *time* *start* *end*) ⇒ void
     extend? : boolean
     time : exact integer
     start : exact non-negative integer or 'start
     end = 'end : exact non-negative integer or 'end

Copies specified range of text into the clipboard. If *extend?* is not #f, the old clipboard contents are appended. If *start* is 'start or *end* is 'end, then the current selection start/end is used.

See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an exn:application:mismatch exception is raised.

- (send *a-text* copy *extend?* *time*) ⇒ void
     extend? = #f : boolean
     time = 0 : exact integer

Copies the selected items into the clipboard. If *extend?* is not #f, the old clipboard contents are appended.

See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an exn:application:mismatch exception is raised.

copy-self-to

Copies the properties of this editor into an existing editor.

- (send *a-text* copy-self-to *dest*) ⇒ void
     dest : text% or pasteboard% object

Each snip in this editor is copied and inserted into *dest*. In addition, this editor's filename, maximum undo history setting, keymap, interactive caret threshold, and overwrte-styles-on-load settings are installed into *dest*. This editor's style list is copied and the copy is installed as the style list for *dest*.

This editor's file format, wordbreak function, wordbreak map, click-between-threshold, caret visbility state, overwrite mode state, and autowrap bitmap are installed into *dest*.

`cut`

Copies and then deletes items in the editor.

The system may execute a cut (in response to other method calls) without calling this method. To extend or re-implement the copying portion of the cut, override the `do-copy` in `text%` or `do-copy` in `pasteboard%` method of an editor. To monitor deletions in an editor, override `on-delete` in `text%` or `on-delete` in `pasteboard%`.

- (send *a-text* `cut` *extend?*   *time start end*) ⇒ void
    extend? : boolean
    time : exact integer
    start : exact non-negative integer or `'start`
    end = `'end` : exact non-negative integer or `'end`

Copies and then deletes the specified range. If *extend?* is not `#f`, the old clipboard contents are appended. If *start* is `'start` or *end* is `'end`, then the current selection start/end is used.

See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

- (send *a-text* `cut` *extend?*   *time*) ⇒ void
    extend? = `#f` : boolean
    time = `0` : exact integer

Copies and then deletes the currently selected items. If *extend?* is not `#f`, the old clipboard contents are appended.

See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`delete`

The content of an editor can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-delete` to monitor content deletion changes.

- (send *a-text* `delete` *start end scroll-ok?*) ⇒ void
    start : exact non-negative integer or `'start`
    end = `'back` : exact non-negative integer or `'back`
    scroll-ok? = `#t` : boolean

Deletes the specified range in the editor. If *start* is `'start`, then the starting selection position is used; if *end* is `'back`, then only the character preceding *start* is deleted. If *scroll-ok?* is not `#f` and *start* is the same as the current caret position, then the editor's display may be scrolled to show the new selection position.

- (send *a-text* `delete`) ⇒ void

Deletes the currently selected text.

`do-copy`

Called to copy a region of the editor into the clipboard. This method is provided so that it can be overriden by subclasses. Do not call this method directly; instead, call `copy`.

- (send *a-text* `do-copy` *start end time extend?*) ⇒ void
    start : exact non-negative integer
    end : exact non-negative integer
    time : exact integer
    extend? : boolean

Copy the data from *start* to *end*, extending the current clipboard contexnts if *extend?* is not `#f`.

See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`do-paste`

Called to paste the current contents of the clipboard into the editor. This method is provided so that it can be overriden by subclasses. Do not call this method directly; instead, call `paste`.

- (send *a-text* `do-paste` *start time*) ⇒ void
    start : exact non-negative integer
    time : exact integer

Paste into the position *start*.

See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`erase`

Erases the contents of the editor.

See also `delete`.

- (send *a-text* `erase`) ⇒ void

`find-line`

Given a graphical location in the editor, returns the line at that location. Lines are numbered starting with 0.

The result is only valid when the editor is displayed (see section 8.1 (page 148)). Calling this method may force the recalculation of location information, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (send *a-text* `find-line` *y on-it?*) ⇒ exact non-negative integer
    y : real number
    on-it? = `#f` : boxed boolean or `#f`

The *on-it?* box is filled with `#t` if the line actually touches this position, or `#f` otherwise, unless *on-it?* is `#f`. (A large enough *y* will always return the last line number, but will set *on-it?* to `#f`.)

`find-position`

Given a graphical location in the editor, returns the item editor position at that location.

The result is only valid when the editor is displayed (see section 8.1 (page 148)).   Calling this method may force the recalculation of location information, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (send *a-text* `find-position` *x* *y* *at-eol?*   *on-it?*   *edge-close?*) ⇒ exact non-negative integer
    x : real number
    y : real number
    at-eol? = `#f` : boxed boolean or `#f`
    on-it? = `#f` : boxed boolean or `#f`
    edge-close? = `#f` : boxed real number or `#f`

See section 8.3 (page 151) for a discussion of the *at-eol?* argument. The *on-it?* box is filled with `#t` if the line actually touches this position, or `#f` otherwise, unless *on-it?* is `#f`.

The *edge-close?* box is filled with it will be filled in with a value indicating how close the point is to the vertical edges of the item when the point falls on the item, unless *edge-close?* is `#f`. If the point is closest to the left edge of the item, the value will be negative; otherwise, the value will be positive. In either case, then absolute value of the returned result is the distance from the point to the edge of the item. The values 100 and -100 indicate infinity.

`find-position-in-line`

Given a graphical location within a line of the editor, returns the item editor position at that location. Lines are numbered starting with `0`.

The result is only valid when the editor is displayed (see section 8.1 (page 148)).   Calling this method may force the recalculation of location information, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (send *a-text* `find-position-in-line` *line* *x* *at-eol?*   *on-it?*   *edge-close?*) ⇒ exact non-negative integer
    line : exact non-negative integer
    x : real number
    at-eol? = `#f` : boxed boolean or `#f`
    on-it? = `#f` : boxed boolean or `#f`
    edge-close? = `#f` : boxed real number or `#f`

See section 8.3 (page 151) for a discussion of the *at-eol?* argument. The *on-it?* box is filled with `#t` if the line actually touches this position, or `#f` otherwise, unless *on-it?* is `#f`.

See `find-position` for a discussion of *edge-close?*.

`find-snip`

Returns the snip at a given position or `#f` if an appropriate snip cannot be found.

- (send *a-text* `find-snip` *pos* *direction* *s-pos*) ⇒ `snip%` object or `#f`
    pos : exact non-negative integer
    direction : symbol in '(`before-or-none before after after-or-none`)
    s-pos = `#f` : boxed exact non-negative integer or `#f`

If the position *pos* is between two snips, *direction* specifies which snip to return; *direction* can be any of the following:

- – 'before-or-none — returns the snip before the position, or #f if *pos* is 0
- – 'before — returns the snip before the position, or the first snip if *pos* is 0
- – 'after — returns the snip after the position, or the last snip if *pos* is the last position
- – 'after-or-none – returns the snip after the position, or #f if *pos* is the last position or larger

The *s-pos* box is filled with the position where the returned snip starts, unless *s-pos* is #f.

find-string

Finds an exact-match string in the editor and returns its position. If the string is not found, #f is returned.

- (send *a-text* find-string *str direction start end get-start? case-sensitive?*)   ⇒ exact non-negative integer or #f
    str : string
    direction = 'forward : symbol in '(forward backward)
    start = 'start : exact non-negative integer or 'start
    end = 'eof : exact non-negative integer or 'eof
    get-start? = #t : boolean
    case-sensitive? = #t : boolean

The *direction* argument can be 'forward or 'backward, indicating a forward search or backward search respectively. In the case of a forward search, the return value is the starting position of the string; for a backward search, the ending position is returned. However, if *get-start?* is #f, then the other end of the string position will be returned.

The *start* and *end* arguments set the starting and ending positions of a forward search (use *start ¿ end* for a backward search). If *start* is 'start, then the search starts at the start of the selection. If *end* is 'eof, then the search continues to the end (for a forward search) or start (for a backward search) of the editor.

If *case-sensitive?* is #f, then an uppercase and lowercase of each alphabetic character are treated as equivalent.

find-string-all

Finds all occurences of a string using find-string. No no occurrences are found, the empty list is returned.

- (send *a-text* find-string-all *str direction start end get-start? case-sensitive*) ⇒ list of exact non-negative integers
    str : string
    direction = 'forward : symbol in '(forward backward)
    start = 'start : exact non-negative integer or 'start
    end = 'eof : exact non-negative integer or 'eof
    get-start? = #t : boolean
    case-sensitive = #t : boolean

The arguments are the same as for find-string.

find-wordbreak

Finds wordbreaks in the editor using the current wordbreak procedure. See also set-wordbreak-func.

- (send *a-text* `find-wordbreak` *start end reason*) ⇒ void
    start : boxed exact non-negative integer or `#f`
    end : boxed exact non-negative integer or `#f`
    reason : symbol in '(caret line selection user1 user2)

The contents of the *start* argument specifies a location to start searching backwards to the next word start; its will be filled with the starting position of the word that is found. If *start* is `#f`, no backward search is performed.

The contents of the *end* argument specifies a location to start searching forwards to the next word end; its will be filled with the ending position of the word that is found. If *end* is `#f`, no forward search is performed.

The *reason* argument specifies more information about what the wordbreak is used for. For example, the wordbreaks used to move the caret may be different from the wordbreaks used to break lines. The possible values of *reason* are:

  − `'caret` — find a wordbreak suitable for moving the caret
  − `'line` — find a wordbreak suitable for breaking lines
  − `'selection` — find a wordbreak suitable for selecting the closest word
  − `'user1` — for other (not built-in) uses
  − `'user2` — for other (not built-in) uses

The actual handling of *reason* is controlled by the current wordbreak procedure; see `set-wordbreak-func`for details. The default handler and default wordbreak map treats alphanumeric charatecters the same for `'caret`, `'line`, and `'selection`. Non-alphanumeric, non-space, non-hypen charaters do not break lines, but do break caret and selection words. For example a comma should not be counted as part of the preceding word for moving the caret past the word or double-clicking the word, but the comma should stay on the same line as the word (and thus counts in the same "line word").

`flash-off`

See `flash-on`. There is no effect if this method is called when flashing is already off.

- (send *a-text* `flash-off`) ⇒ void

    Turns off the hiliting and shows the normal selection range again.

`flash-on`

Temporarily hilites a region in the editor without changing the current selection.

- (send *a-text* `flash-on` *start end at-eol? scroll? timeout*) ⇒ void
    start : exact non-negative integer
    end : exact non-negative integer
    at-eol? = `#f` : boolean
    scroll? = `#t` : boolean
    timeout = 500 : exact non-negative integer

See section 8.3 (page 151) for a discussion of the *at-eol?* argument. If *scroll?* is not `#f`, the editor display will be scrolled if necessary to show the hilited region. If *timeout* is greater than 0, then the hiliting will be automatically turned off after the given number of milliseconds.

See also `flash-off`.

get-anchor

Returns `#t` if the selection is currently auto-extending.

- (send *a-text* `get-anchor`) ⇒ boolean

get-between-threshold

Returns an amount used to determine the meaning of a user click. If the click falls within the threshold of a positon between two items, then the click registers on the space between the items rather than on either item.

See also `set-between-threshold`.

- (send *a-text* `get-between-threshold`) ⇒ non-negative real number

get-character

Gets a single character for the editor.

- (send *a-text* `get-character` *start*) ⇒ character
    start : exact non-negative integer

  Returns the character following the position *start*. If *start* is greater than or equal to the last position, the null character is returned.

get-end-position

Returns the ending position of the current selection. See also `get-position`.

- (send *a-text* `get-end-position`) ⇒ exact non-negative integer

get-file-format

Returns the format of the last file saved from or loaded into this editor. See also `load-file`.

- (send *a-text* `get-file-format`) ⇒ symbol in '(standard text text-force-cr)

get-line-spacing

Returns the spacing inserted by the editor between each line. This spacing is included in the reported height of each line.

- (send *a-text* `get-line-spacing`) ⇒ non-negative real number

get-overwrite-mode

Returns `#t` if the editor is in overwrite mode, `#f` otherwise. Overwrite mode only affects the way that `on-default-char`handles keyboard input for insertion characters. See also `set-overwrite-mode`.

- (send *a-text* `get-overwrite-mode`) ⇒ boolean

get-position

Returns the current selection range. See also `get-start-position` and `get-end-position`.

- (send *a-text* `get-position` *start end*) ⇒ void
    start : boxed exact non-negative integer or `#f`
    end = `#f` : boxed exact non-negative integer or `#f`

    The *start* box is filled with the starting position of the selection, unless *start* is `#f`. The *end* box is filled with the ending position of the selection, unless *end* is `#f`.

get-region-data

Gets extra data associated with a given region. See section 8.2.1 (page 151)for more information. This method is *not* called when the whole editor is saved to a file; in such cases, the information can be stored in the header or footer.

- (send *a-text* `get-region-data` *start end*) ⇒ `editor-data%` object or `#f`
    start : exact non-negative integer
    end : exact non-negative integer

get-snip-position

Returns the starting item position of a given snip or `#f` if the snip is not in this editor.

- (send *a-text* `get-snip-position` *snip*) ⇒ exact non-negative integer or `#f`
    snip : `snip%` object

get-snip-position-and-location

Gets a snip's item position and top left display location in editor coordinates. The return value is `#t` if the snip is found, `#f` otherwise.

When location information is requested: The result is only valid when the editor is displayed (see section 8.1 (page 148)). Calling this method may force the recalculation of location information, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (send *a-text* `get-snip-position-and-location` *snip pos x y*) ⇒ boolean
    snip : `snip%` object
    pos : boxed exact non-negative integer or `#f`
    x = `#f` : boxed real number or `#f`
    y = `#f` : boxed real number or `#f`

    The *pos* box is filled with starting position of *snip*, unless *pos* is `#f`. The *x* box is filled with left location of *snip* in editor coordinates, unless *x* is `#f`. The *y* box is filled with top location of *snip* in editor coordinates, unless *y* is `#f`.

get-start-position

Returns the starting position of the current selection. See also `get-position`.

- (send *a-text* get-start-position) $\Rightarrow$ exact non-negative integer

`get-styles-sticky`

In the normal mode for a text editor, style settings are sticky. With sticky styles, when a string or character is inserted into an editor, it gets the style of the snip preceding the insertion point (or the snip that includes the insertion point if text is inserted into an exiting string snip). Alternatively, if `change-style` is called to set the style at the caret position (when it is not a range), then the style is remembered; if the editor is not changed before text is inserted at the caret, then the text gets the remembered style.

With non-sticky styles, text inserted into an editor always gets the style named "Standard" in the editor's style list.

See also `set-styles-sticky`.

- (send *a-text* get-styles-sticky) $\Rightarrow$ boolean

`get-tabs`

Returns the current tab position array as a list.

- (send *a-text* get-tabs *length tab-width in-units*) $\Rightarrow$ list of real numbers
  length = `#f` : boxed exact non-negative integer or `#f`
  tab-width = `#f` : boxed real number or `#f`
  in-units = `#f` : boxed boolean or `#f`

  The *length* box is filled with the length of the tab array (and therefore the returned list), unless *length* is `#f`. The *tab-width* box is filled with the width used for tabs past the end of the tab array, unless *tab-width* is `#f`. The *in-units* box is filled with `#t` if the tabs are specified in canvas units or `#f` if they are specified in space-widths, unless *in-units* is `#f`.

  See also `set-tabs`.

`get-text`

Returns the contents of the editor in text form.

- (send *a-text* get-text *start end flattened? force-cr?*) $\Rightarrow$ string
  start = 0 : exact non-negative integer
  end = `'eof` : exact non-negative integer or `'eof`
  flattened? = `#f` : boolean
  force-cr? = `#f` : boolean

  Gets the text from *start* to *end*. If *end* is `'eof`, then the contents are returned from *start* until the end of the editor.

  If *flattened?* is not `#f`, then flattened text is returned. See section 8.4 (page 152) for a discussion of flattened vs. non-flattened text.

  If *force-cr?* is not `#f` and *flattened?* is not `#f`, then automatic carriage returns (from word-wrapping) are written into the return string as real carriage returns.

`get-top-line-base`

Returns the distance from the top of the editor to the alignment baseline of the top line. This method is primarily used when an editor is an item within another editor.

The result is only valid when the editor is displayed (see section 8.1 (page 148)). For `text%` objects, calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (send *a-text* get-top-line-base) ⇒ non-negative real number

`get-visible-line-range`

Returns the range of lines which are currently visible to the user. Lines are numbered starting with `0`.

The result is only valid when the editor is displayed (see section 8.1 (page 148)). Calling this method may force the recalculation of location information, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (send *a-text* get-visible-line-range *start* *end*) ⇒ void
    start : boxed exact non-negative integer or #f
    end : boxed exact non-negative integer or #f

The *start* box is filled with first line visible to the user, unless *start* is #f. The *end* box is filled with last line visible to the user, unless *end* is #f.

`get-visible-position-range`

Returns the range of positions which are currently visible to the user.

The result is only valid when the editor is displayed (see section 8.1 (page 148)). Calling this method may force the recalculation of location information, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (send *a-text* get-visible-position-range *start* *end*) ⇒ void
    start : boxed exact non-negative integer or #f
    end : boxed exact non-negative integer or #f

The *start* box is filled with first position visible to the user, unless *start* is #f. The *end* box is filled with last position visible to the user, unless *end* is #f.

`get-wordbreak-map`

Returns the wordbreaking map that is used by the standard wordbreaking function. See `editor-wordbreak-map%` for more information.

- (send *a-text* get-wordbreak-map) ⇒ editor-wordbreak-map% object

`hide-caret`

Determines whether the caret is shown when the editor has the keyboard focus.

277

- (send *a-text* `hide-caret` *hide?*) ⇒ void
    hide? : boolean

If *hide?* is not `#f`, then the caret or selection hiliting will not be drawn for the editor. The editor can still own the keyboard focus, but no caret will be drawn to indicate the focus.

See also `caret-hidden?`and `lock`.

`insert`

Inserts data into the editor.

The content of an editor can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-insert` in `text%` or `on-insert` in `pasteboard%` to monitor content additions changes.

- (send *a-text* `insert` *str start end scroll-ok?*) ⇒ void
    str : string
    start : exact non-negative integer
    end = `'same` : exact non-negative integer or `'same`
    scroll-ok? = `#t` : boolean

Inserts the text *str* at position *start*.

If *end* is not `'same`, then *str* replaces the region from *start* to *end*, and the selection is left at the end of the inserted text. Otherwise, If the insertion position is before or equal to the selection's start/end position, then the selection's start/end position is incremented by the length of *str*.

If *scroll-ok?* is not `#f` and *start* is the same as the current selection's start position, then the editor's display is scrolled to show the new selection position.

See also `get-styles-sticky`.

- (send *a-text* `insert` *n str start end scroll-ok?*) ⇒ void
    n : exact non-negative integer
    str : string
    start : exact non-negative integer
    end = `'same` : exact non-negative integer or `'same`
    scroll-ok? = `#t` : boolean

Inserts the first *n* characters of *str* at position *start*.

If *end* is not `'same`, then the inserted text replaces the region from *start* to *end*, and the selection is left at the end of the inserted text. Otherwise, If the insertion position is before or equal to the selection's start/end position, then the selection's start/end position is incremented by *n*.

If *scroll-ok?* is not `#f` and *start* is the same as the current select's start position, then the editor's display is scrolled to show the new selection position.

See also `get-styles-sticky`.

- (send *a-text* `insert` *str*) ⇒ void
    str : string

Inserts *str* at the current selection start position.

If the current selection covers a range of items, then *str* replaces the selected text. The selection's starts and end positions are moved to the end of the inserted text.

The editor's display is scrolled to show the new selection position.

See also `get-styles-sticky`.

- (send *a-text* `insert` *n* *str*) ⇒ void
    n : exact non-negative integer
    str : string

Inserts the first *n* characters of *str* at the current selection start position.

If the current selection covers a range of items, then the inserted text replaces the selected text. The selection's start and end positions are moved to the end of the inserted text.

The editor's display is scrolled to show the new selection position.

See also `get-styles-sticky`.

- (send *a-text* `insert` *snip* *start* *end* *scroll-ok?*) ⇒ void
    snip : `snip%` object
    start : exact non-negative integer
    end = `'same` : exact non-negative integer or `'same`
    scroll-ok? = `#t` : boolean

Inserts *snip* into the editor at *start*. A snip cannot be inserted into multiple editors or multiple times within a single editor.

If *end* is not `'same`, then *snip* replaces the region from *start* to *end*, and the selection is left at the end of the inserted snip. Otherwise, If the insertion position is before or equal to the selection's start/end position, then the selection's start/end position is incremented by the item length of *snip*.

If *scroll-ok?* is not `#f` and *start* is the same as the current selection's start position, then the editor's display is scrolled to show the new selection position.

See also `get-styles-sticky`.

- (send *a-text* `insert` *snip*) ⇒ void
    snip : `snip%` object

Inserts *snip* into the editor at the current selection position. A snip cannot be inserted into multiple editors or multiple times within a single editor.

If the current selection covers a range of items, then the inserted text replaces the selected text. The selection's start and end positions are moved to the end of the inserted snip.

The editor's display is scrolled to show the new selection position.

- (send *a-text* `insert` *char*) ⇒ void
    char : character

Inserts *char* into the editor at the current selection position.

If the current selection covers a range of items, then *char* replaces the selected text. The selection's start and end positions are moved to the end of the inserted character.

The editor's display is scrolled to show the new selection position.

See also `get-styles-sticky`.

- (send *a-text* `insert` *char* *start* *end*) ⇒ void
    char : character
    start : exact non-negative integer
    end = `'same` : exact non-negative integer or `'same`

Inserts *char* into the editor at the position *start*.

If *end* is not `'same`, then *char* replaces the region from *start* to *end*, and the selection is left at the end of the inserted text. Otherwise, If the insertion position is before or equal to the selection's start/end position, then the selection's start/end position is incremented by `1`.

If *start* is the same as the current selection's start position, then the editor's display is scrolled to show the new selection position.

See also `get-styles-sticky`.

`kill`

In a text editor, cuts to the end of the current line, or cuts a newline if there is only whitespace between the selection and end of line. Multiple consective kills are appended. In a pasteboard editor, cuts the current selection.

See also `cut`.

The content of an editor can be changed by the system in response to other method calls, and such changes do not go through this method; use `on-delete` in `text%` or `on-delete` in `pasteboard%` to monitor content deletions changes.

- (**send** *a-text* `kill` *time* *start* *end*) ⇒ void
  time : exact integer
  start : exact non-negative integer
  end : exact non-negative integer

  Cuts the text in the given region.

  See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

- (**send** *a-text* `kill` *time*) ⇒ void
  time = 0 : exact integer

  See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

`last-line`

Returns the number of the last line in the editor. Lines are numbered starting with `0`, so this is one less than the number of lines in the editor.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

See also `paragraph-start-position`, which operates on paragraphs (determined by explicit newline characters) instead of lines (determined by both explicit newline characters and automatic line-wrapping).

- (**send** *a-text* `last-line`) ⇒ exact non-negative integer

`last-paragraph`

Returns the number of the last paragraph in the editor. Paragraphs are numbered starting with `0`, so this is one less than the number of paragraphs in the editor.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (**send** *a-text* `last-paragraph`) ⇒ exact non-negative integer

`last-position`

Returns the last selection position in the editor. This is also the number of items in the editor.

- (send *a-text* `last-position`) ⇒ exact non-negative integer

`line-end-position`

Returns the last position of a given line. Lines are numbered starting with `0`.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

See also `paragraph-start-position`, which operates on paragraphs (determined by explicit newline characters) instead of lines (determined by both explicit newline characters and automatic line-wrapping).

- (send *a-text* `line-end-position` *line visible?*) ⇒ exact non-negative integer
    line : exact non-negative integer
    visible? = `#t` : boolean

   If there are fewer than *line*−1 lines, the end of the last line is returned. If *line* is less than 0, then the end of the first line is returned.

   If the line ends with invisible items (such as a carriage return) and *visible?* is not `#f`, the first position before the invisible items is returned.

`line-length`

Returns the number of items in a given line. Lines are numbered starting with `0`.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

- (send *a-text* `line-length` *i*) ⇒ exact non-negative integer
    i : exact non-negative integer

`line-location`

Given a line number, returns the graphic location of the line. Lines are numbered starting with `0`.

The result is only valid when the editor is displayed (see section 8.1 (page 148)). Calling this method may force the recalculation of location information, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

See also `paragraph-start-position`, which operates on paragraphs (determined by explicit newline characters) instead of lines (determined by both explicit newline characters and automatic line-wrapping).

- (send *a-text* `line-location` *line top?*) ⇒ real number
    line : exact non-negative integer
    top? = `#t` : boolean

   If *top?* is not `#f`, the location for the top of the line is returned; otherwise, the the location for the bottom of the line is returned.

`line-paragraph`

Returns the paragraph number of the paragraph containing the line. Lines are numbered starting with 0. Paragraphs are numbered starting with 0.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

- (send *a-text* `line-paragraph` *start*) ⇒ exact non-negative integer
  start : exact non-negative integer

`line-start-position`

Returns the first position of the given line. Lines are numbered starting with 0.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

To calculate lines, if the following are true:

- the editor is not displayed (see section 8.1 (page 148)),

- a maximum width is set for the editor, and

- the editor has never been viewed

then this method ignores the editor's maximum width and any automatic line breaks it might imply. If the first two of the above conditions are true and the editor was *formerley* displayed, this method uses the line breaks from the most recent display of the editor. (Insertions or deletions since the display shift line breaks within the editor in the same way as items.)

See also `paragraph-start-position`, which operates on paragraphs (determined by explicit newline characters) instead of lines (determined by both explicit newline characters and automatic line-wrapping).

- (send *a-text* `line-start-position` *line visible?*) ⇒ exact non-negative integer
  line : exact non-negative integer
  visible? = `#t` : boolean

  If there are fewer than *line*−1 lines, the start of the last line is returned. If *line* is less than 0, then the start of the first line is returned.

  If the line starts with invisible items and *visible?* is not `#f`, the first position past the invisible items is returned.

`move-position`

Move the current selection.

See also `set-position`.

- (send *a-text* `move-position` *code extend? kind*) ⇒ void
  code : symbol in '(`home end right left up down`)

    extend? = #f : boolean
    kind = 'simple : symbol in '(simple word page line)

The possible values for *code* are:

- 'home — go to start of file
- 'end — go to end of file
- 'right — move right
- 'left — move left
- 'up — move up
- 'down — move down

If *extend?* is not #f, the selection range is extended instead of moved.

The possible values for *kind* are:

- 'simple — move one item or line
- 'word — works with 'right or 'left
- 'page — works with 'up or 'down
- 'line — works with 'right or 'left; moves to the start or end of the line

on-change-style

Called before the style is changed in a given range of the editor, after `can-change-style?` is called to verify that the change is ok. The `after-change-style` method is guaranteed to be called after the change has completed.

The editor is internally locked for writing during a call to this method (see also section 8.8 (page 153)). Use `after-change-style` to modify the editor, if necessary.

See also `on-edit-sequence`.

- (send *a-text* on-change-style *start len*) ⇒ void
    start : exact non-negative integer
    len : exact non-negative integer

on-default-char

Called by `on-local-char` when the event is *not* handled by a caret-owning snip or by the keymap.

- (send *a-text* on-default-char *event*) ⇒ void
    event : key-event% object

Handles the following:

- Delete and Backspace — calls `delete`.
- The arrow keys, Page Up, Page Down, Home, and End (including shfted versions) — moves the selection position with `move-position`.
- Any other character in the range (integer->char 32) to (integer->char 255) — inserts the character into the editor.

on-delete

Called before a range is deleted from the editor, after `can-delete?` is called to verify that the deletion is ok. The `after-delete` method is guaranteed to be called after the delete has completed.

The editor is internally locked for writing during a call to this method (see also section 8.8 (page 153)). Use `after-delete` to modify the editor, if necessary.

See also `on-edit-sequence`.

- (send *a-text* `on-delete` *start len*) $\Rightarrow$ void
    start : exact non-negative integer
    len : exact non-negative integer

The *start* argument specifies the starting position of the range to delete. The *len* argument specifies number of items to delete (so *start* + *length* is the endig position of the range to delete).

on-insert

Called before items are inserted into the editor, after `can-insert?` is called to verify that the insertion is ok. The `after-insert` method is guaranteed to be called after the insert has completed.

The editor is internally locked for writing during a call to this method (see also section 8.8 (page 153)). Use `after-insert` to modify the editor, if necessary.

See also `on-edit-sequence`.

- (send *a-text* `on-insert` *start len*) $\Rightarrow$ void
    start : exact non-negative integer
    len : exact non-negative integer

The *start* argument specifies the position of the insert. The *len* argument specifies the total length (in positions) of the items to be inserted.

on-new-string-snip

Creates and returns a new instance of `string-snip%` to store inserted text.

- (send *a-text* `on-new-string-snip`) $\Rightarrow$ `string-snip%` object

on-new-tab-snip

Creates and returns a new instance of `tab-snip%` to store an inserted tab.

- (send *a-text* `on-new-tab-snip`) $\Rightarrow$ `tab-snip%` object

on-set-size-constraint

Called before the editor's maximum or minimum height or width is changed, after `can-set-size-constraint?` is called to verify that the change is ok. The `after-set-size-constraint` method is guaranteed to be called after the change has completed.

(This callback method is provided because setting an editor's maxmimum width may cause lines to be re-flowed with soft carriage returns.)

See also `on-edit-sequence`.

- (send *a-text* `on-set-size-constraint`) $\Rightarrow$ void

`paragraph-end-line`

Returns the ending line of a given paragraph. Paragraphs are numbered starting with `0`. Lines are numbered starting with `0`.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

- (send *a-text* `paragraph-end-line` *paragraph*) $\Rightarrow$ exact non-negative integer
  paragraph : exact non-negative integer

`paragraph-end-position`

Returns the ending position of a given paragraph. Paragraphs are numbered starting with `0`.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

- (send *a-text* `paragraph-end-position` *paragraph* *visible?*) $\Rightarrow$ exact non-negative integer
  paragraph : exact non-negative integer
  visible? = `#f` : boolean

If there are fewer than *paragraph*-1 paragraphs, the end of the last paragraph is returned. If *paragraph* is less than 0, then the end of the first paragraph is returned.

If the paragraph ends with invisible items (such as a carriage return) and *visible?* is not `#f`, the first position before the invisible items is returned.

`paragraph-start-line`

Returns the starting line of a given paragraph. Paragraphs are numbered starting with `0`. Lines are numbered starting with `0`.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

- (send *a-text* `paragraph-start-line` *paragraph*) $\Rightarrow$ exact non-negative integer
  paragraph : exact non-negative integer

`paragraph-start-position`

Returns the starting position of a given paragraph. Paragraphs are numbered starting with `0`.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

- (send *a-text* `paragraph-start-position` *paragraph* *visible?*) $\Rightarrow$ exact non-negative integer
  paragraph : exact non-negative integer
  visible? = `#f` : boolean

If there are fewer than *paragraph*−1 paragraphs, the start of the last paragraph is returned.

If the paragraph starts with invisible items and *visible?* is not `#f`, the first position past the invisible items is returned.

paste

Pastes the current contents of the clipboard into the editor.

The system may execute a paste (in response to other method calls) without calling this method. To extend or re-implement copying, override the `do-paste` in `text%` or `do-paste` in `pasteboard%` method of an editor.

See also `get-paste-text-only` .

- (send *a-text* `paste` *time start end*) ⇒ void
    time : exact integer
    start : exact non-negative integer or `'end`
    end = `'same` : exact non-negative integer or `'same`

  Pastes into the specified range. If *start* is `'end`, then the current selection end position is used. If *end* is `'same`, then *start* is used for *end*.

  See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

- (send *a-text* `paste` *time*) ⇒ void
    time = 0 : exact integer

  See section 8.6 (page 152) for a discussion of the *time* argument. If *time* is outside the platform-specific range of times, an `exn:application:mismatch` exception is raised.

paste-next

Editors collectively manitain a copy ring that holds up to 30 previous copies (and cuts) among the editors. When it is called as the next method on an editor after a paste, the `paste-next` method replaces the text from a previous paste with the next data in the copy ring, incrementing the ring pointer so that the next `paste-next` pastes in even older data.

It is a copy "ring" because the ring pointer wraps back to the most recent copied data after the oldest remembered data is pasted. Any cut, copy, or (regular) paste operation resets the copy ring pointer back to the beginning.

If the previous operation on the editor was not a paste, calling `paste-next` has no effect.

- (send *a-text* `paste-next`) ⇒ void

position-line

Returns the line number of the line containing a given position. Lines are numbered starting with `0`.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

See also `paragraph-start-position`, which operates on paragraphs (determined by explicit newline characters) instead of lines (determined by both explicit newline characters and automatic line-wrapping).

- (send *a-text* `position-line` *start at-eol?*) ⇒ exact non-negative integer
   start : exact non-negative integer
   at-eol? = `#f` : boolean

See section 8.3 (page 151) for a discussion of *at-eol?*.

`position-location`

Returns the graphic locaiton of a given position.

The result is only valid when the editor is displayed (see section 8.1 (page 148)).   Calling this method may force the recalculation of location information, even if the editor currently has delayed refreshing (see `refresh-delayed?`).

- (send *a-text* `position-location` *start x y front?   at-eol?   whole-line?*) ⇒ void
   start : exact non-negative integer
   x = `#f` : boxed real number or `#f`
   y = `#f` : boxed real number or `#f`
   front? = `#t` : boolean
   at-eol? = `#f` : boolean
   whole-line? = `#f` : boolean

The *x* box is filled with the x-location of the position *start* in editor coordinates, unless *x* is `#f`. The *y* box is filled with the y-location (top or bottom; see below) of the position *start* in editor coordinates, unless *y* is `#f`.

See section 8.3 (page 151) for a discussion of *at-eol?*.

If *front?* is not `#f`, the top coordinate of the location is returned, otherwise the bottom coordinate of the location is returned.

The top *y* location may be different for different positions within a line when different-sized graphic objects are used. If *whole-line?* is not `#f`, the minimum top location or maximum bottom location for the whole line is returned in *y*.

`position-paragraph`

Returns the paragraph number of the paragraph containing a given position.

Calling this method may force the recalculation of location information if a maximum width is set for the editor, even if the editor currently has delayed refreshing (see `refresh-delayed?`). If the editor is not displayed and the editor has a maximum width, line breaks are calculated as for `line-start-position`.

- (send *a-text* `position-paragraph` *start at-eol?*) ⇒ exact non-negative integer
   start : exact non-negative integer
   at-eol? = `#f` : boolean

See section 8.3 (page 151) for a discussion of *at-eol?*.

`read-from-file`

Reads new contents for the editor from a stream. The return value is `#t` if there are no errors, `#f` otherwise. See also section 8.2 (page 150).

The stream provides either new mappings for names in the editor's style list, or it indicates that the editor should share a previously-read style list (depending on how style lists were shared when the editor was written to the stream; see also `write-to-file`).

- In the former case, if the *overwrite-styles?* argument is is `#f`, then each style name in the loaded file that is already in the current style list keeps its current style. Otherwise, existing named styles are overwritten with specifications from the loaded file.

- In the latter case, the editor's style list will be changed to the previously-read list.

- (send *a-text* `read-from-file` *stream start overwrite-styles?*) ⇒ boolean
    stream : `editor-stream-in%` object
    start : exact non-negative integer or `'start`
    overwrite-styles? = `#t` : boolean

    New data is inserted at the position indicated by *start*, or at the current position if *start* is `'start`.

- (send *a-text* `read-from-file` *stream overwrite-styles?*) ⇒ boolean
    stream : `editor-stream-in%` object
    overwrite-styles? = `#t` : boolean

`remove-clickback`

Removes clickbacks. See also section 8.7 (page 153).

- (send *a-text* `remove-clickback` *start end*) ⇒ void
    start : exact non-negative integer
    end : exact non-negative integer

    Removes all clickbacks installed for exacty the range *start* to *end*.

`scroll-to-position`

Scrolls the editor so that a given position is visible.

Scrolling is disallowed when the editor is internally locked for reflowing (see also section 8.8 (page 153)).

The system may scroll the editor without calling this method.

- (send *a-text* `scroll-to-position` *start at-eol?*   *end bias*) ⇒ boolean
    start : exact non-negative integer
    at-eol? = `#f` : boolean
    end = `'same` : exact non-negative integer or `'same`
    bias = `'none` : symbol in `'(start end none)`

    If *end* is `'same` or equal to *start*, then position *start* is made visible. See section 8.3 (page 151) for a discussion of *at-eol?*.

    If *end* is not `'same` and not the same as *start*, then the range *start* to *end* is made visible and *at-eol?* is ignored.

    When the specified range cannot fit in the visible area, *bias* indicates which end of the range to display. When *bias* is `'same`, then the start of the range is displayed. When *bias* is `'end`, then the end of the range is displayed. Otherwise, *bias* must be `'none`.

    If the editor is scrolled, then the editor is redrawn and the return value is `#t`; otherwise, the return value is `#f`.

set-anchor

Turns anchoring on or off. This method can be overridden to affect or detect changes in the anchor state.

- (send *a-text* `set-anchor` *on?*) ⇒ void
    on? : boolean

    If *on?* is not `#f`, then the selection will be automatically extended when cursor keys are used, otherwise anchoring is turned off. Anchoring is automatically turned off if the user does anything besides cursor movements.

set-autowrap-bitmap

Sets the bitmap that is drawn at the end of a line when it is automatically line-wrapped.

The bitmap will not be modified. It may be selected into a `bitmap-dc%` object, but it will be selected out if this method is called again.

Setting the bitmap is disallowed when the editor is internally locked for reflowing (see also section 8.8 (page 153)).

- (send *a-text* `set-autowrap-bitmap` *bitmap*) ⇒ `bitmap%` object or `#f`
    bitmap : `bitmap%` object or `#f`

    If *bitmap* is `#f`, no autowrap indicator is drawn (this is the default). The previously used bitmap (possibly `#f`) is returned.

set-between-threshold

Sets the graphical distance used to determine the meaning of a user click.

- (send *a-text* `set-between-threshold` *threshold*) ⇒ void
    threshold : non-negative real number

    If the click falls within *threshold* of a positon between two items, then the click registers on the space between the items rather than on either item.

    See also `get-between-threshold`.

set-clickback

Installs a clickback for a given region. See also section 8.7 (page 153).

- (send *a-text* `set-clickback` *start* *end* *f* *hilite-delta* *call-on-down?*) ⇒ void
    start : exact non-negative integer
    end : exact non-negative integer
    f : procedure of three arguments: a `text%` object, a starting position exact non-negative
        integer, and an ending position exact non-negative integer
    hilite-delta = `#f` : `style-delta%` object or `#f`
    call-on-down? = `#f` : boolean

    The callback procedure *f* is called when the user selects the clickback. The arguments to *f* are this editor and the starting and ending range of the clickback.

The *hilite-delta* style delta is applied to the clickback text when the user has clicked and is still holding the mouse over the clickback. If *hilite-delta* is #f, then the clickback region's style is not changed when it is being selected.

If *call-on-down?* is not #f, the clickback is called immediately when the user clicks the mouse button down, instead of after a mouse-up event. The *hilite-delta* argument is not used in this case.

set-file-format

Set the format of the file saved from this editor.

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

- (send *a-text* set-file-format *format*) ⇒ void
    format : symbol in '(standard text text-force-cr)

  The legal formats are:

  – '`standard` — a standard editor file
  – '`text` — a text file
  – '`text-force-cr` — a text file; when writing, change automatic newlines (from word-wrapping) into real carriage returns

set-line-spacing

Sets the spacing inserted by the editor between each line. This spacing is included in the reported height of each line.

- (send *a-text* set-line-spacing *space*) ⇒ void
    space : non-negative real number

set-overwrite-mode

Enables or disables overwrite mode. See `get-overwrite-mode`. This method can be overridden to affect or detect changes in the overwrite mode.

- (send *a-text* set-overwrite-mode *on?*) ⇒ void
    on? : boolean

set-paragraph-alignment

Sets a paragraph-specific horizontal alignment. The alignment is only used when the editor has a maximum width, as set with `set-max-width`. Paragraphs are numbered starting with 0.

This method is experimental, and works reliably only when the paragraph is not merged or split. Merging or splitting a paragraph with alignment settings causes the settings to be transfered unpredictably (although other paragraphs in the editor can be safely split or merged). If the last paragraph in an editor is empty, settings assigned to it are ignored.

- (send *a-text* set-paragraph-alignment *paragraph alignment*) ⇒ void
    paragraph : exact non-negative integer
    alignment : symbol in '(left center right)

`set-paragraph-margins`

Sets a paragraph-specific margin. Paragraphs are numbered starting with 0.

This method is experimental, and works reliably only when the paragraph is not merged or split. Merging or splitting a paragraph with margin settings causes the settings to be transfered unpredictably (although other paragraphs in the editor can be safely split or merged). If the last paragraph in an editor is empty, settings assigned to it are ignored.

- (send *a-text* `set-paragraph-margins` *paragraph first-left left right*) ⇒ void
    paragraph : exact non-negative integer
    first-left : non-negative real number
    left : non-negative real number
    right : non-negative real number

   The first line of the paragraph is indented by *first-left* points within the editor. If the paragraph is line-wrapped (when the editor has a maximum width), subsequent lines are indented by *left* points. If the editor has a maximum width, the paragraph's maximum width for line-wrapping is *right* points smaller than the editor's maximum width.

`set-position`

Sets the current selection in the editor.

Setting the position is disallowed when the editor is internally locked for reflowing (see also section 8.8 (page 153)).

The system may change the selection in an editor without calling this method (or any visible method).

- (send *a-text* `set-position` *start end at-eol? scroll? seltype*) ⇒ void
    start : exact non-negative integer
    end = `'same` : exact non-negative integer or `'same`
    at-eol? = `#f` : boolean
    scroll? = `#t` : boolean
    seltype = `'default` : symbol in `'(default x local)`

   If *end* is `'same` or less than or equal to *start*, the current start and end positions are both set to *start*. Otherwise the given range is selected.

   See section 8.3 (page 151) for a discussion of *at-eol?*. If *scroll?* is not `#f`, then the display is scrolled to show the selection if necessary.

   The *seltype* argument is only used when the X Window System selection mechanism is enabled. The possible values are:

   – `'default` — if this window has the keyboard focus and given selection is non-empty, make it the current X selection
   – `'x` — if the given selection is non-empty, make it the current X selection
   – `'local` — do not change the current X selection

   See also `editor-set-x-selection-mode`.

`set-position-bias-scroll`

Like `set-position`, but a scrolling bias can be specified.

291

- (send *a-text* set-position-bias-scroll *bias start end ateol? scroll? seltype*) ⇒ void
  bias : symbol in '(start-only start none end end-only)
  start : exact non-negative integer
  end = 'same : exact non-negative integer or 'same
  ateol? = #f : boolean
  scroll? = #t : boolean
  seltype = 'default : symbol in '(default x local)

The possible values for *bias* are:

 – 'start-only — only insure that the starting position is visible
 – 'start — if the range doesn't fit in the visible area, show the starting position
 – 'none — no special scrolling instructions
 – 'end — if the range doesn't fit in the visible area, show the ending position
 – 'end-only — only insure that the ending position is visible

See also `scroll-to-position`.

set-region-data

Sets extra data associated with a given region. See section 8.2.1 (page 151)and `get-region-data` for more information.

- (send *a-text* set-region-data *start end data*) ⇒ void
  start : exact non-negative integer
  end : exact non-negative integer
  data : editor-data% object

set-styles-sticky

See `get-styles-sticky` for information about sticky styles.

- (send *a-text* set-styles-sticky *sticky?*) ⇒ void
  sticky? : boolean

set-tabs

Sets the tabbing array for the editor.

Setting tabs is disallowed when the editor is internally locked for reflowing (see also section 8.8 (page 153)).

- (send *a-text* set-tabs *tabs tab-width in-units?*) ⇒ void
  tabs : list of real numbers
  tab-width = 20 : real number
  in-units? = #t : boolean

The *tabs* list determines the tabbing array. The tabbing array specifies the x-locations where each tab occurs. Tabs beyond the last specified tab are separated by a fixed amount *tab-width*. If *in-units?* is not #f, then tabs are specified in canvas units; otherwise, they are specified as a number of spaces. (If tabs are specified in spaces, then the graphic tab positions will change with the font used for the tab.)

`set-wordbreak-func`

Sets the word-breaking function for the editor. For information about the arguments to the word-breaking function, see `find-wordbreak`.

The standard wordbreaking function uses the editor's `editor-wordbreak-map%` object to determine which characters break a word. See also `editor-wordbreak-map%` and `set-wordbreak-map`.

Since the wordbreak function will be called when line breaks are being determined (in an editor that has a maximum width), there is a constrained set of `text%` methods that the wordbreak function is allowed to invoke. It cannot invoke a member function that uses information about graphic locations or lines (which are identified in this manual with " The result is only valid when the editor is displayed (see section 8.1 (page 148))."), but it can still invoke member functions that work with snips and item positions.

- (send *a-text* `set-wordbreak-func` *f*) ⇒ void
    f : procedure of four arguments: a `text%` object, a boxed exact non-negative integer or
        `#f`, another boxed exact non-negative integer or `#f`, and a symbol

`set-wordbreak-map`

Sets the wordbreaking map that is used by the standard wordbreaking function. See `editor-wordbreak-map%` for more information.

- (send *a-text* `set-wordbreak-map` *map*) ⇒ void
    map : `editor-wordbreak-map%` object or `#f`

    If *map* is `#f`, then then standard map (`the-editor-wordbreak-map`) is used.

`split-snip`

Given a position, splits a snip that includes the position so that the position is between two snips. The snip may refuse to split, although none of the built-in snip classes will ever refuse.

Splitting a snip is disallowed when the editor is internally locked for reflowing (see also section 8.8 (page 153)).

- (send *a-text* `split-snip` *pos*) ⇒ void
    pos : exact non-negative integer

`write-to-file`

Writes the current editor contents to the given stream. The return value is `#t` if there are no errors, `#f` otherwise. See also section 8.2 (page 150).

If the editor's style list has already been written to the stream, it is not re-written. Instead, the editor content indicates that the editor shares a previously-written style list. This sharing will be recreated when the stream is later read.

- (send *a-text* `write-to-file` *stream* *start* *end*) ⇒ boolean
    stream : `editor-stream-out%` object
    start : exact non-negative integer
    end = `'eof` : exact non-negative integer or `'eof`

If *start* is 0 and *end* is `'eof` negative, then the entire contents are written to the stream. If *end* is `'eof`, then the contents are written from *start* until the end of the editor. Otherwise, the contents of the given range are written.

- (`send` *a-text* `write-to-file` *stream*) ⇒ boolean
  stream : `editor-stream-out%` object

10. Editor Procedures

10.1 Editors

`add-editor-keymap-functions`

Given a `keymap%` object, the keymap is loaded with mappable functions that apply to all `editor<%>` objects:

- "copy-clipboard"
- "copy-append-clipboard"
- "cut-clipboard"
- "cut-append-clipboard"
- "paste-clipboard"
- "delete-selection"
- "clear-selection"
- "undo"
- "redo"
- "select-all"

- (add-editor-keymap-functions *keymap*) ⇒ void
    keymap : `keymap%` object

`add-pasteboard-keymap-functions`

Given a `keymap%` object, the table is loaded with mappable functions that apply to `pasteboard%` objects. Currently, there are no such functions.

See also `add-editor-keymap-functions`.

- (add-pasteboard-keymap-functions *keymap*) ⇒ void
    keymap : `keymap%` object

`add-text-keymap-functions`

Given a `keymap%` object, the table is loaded with functions that apply to all `text%` objects:

- "forward-character"
- "backward-character"
- "previous-line"
- "next-line"
- "previous-page"
- "next-page"
- "forward-word"

- "backward-word"
- "forward-select"
- "backward-select"
- "select-down"
- "select-up"
- "select-page-up"
- "select-page-down"
- "forward-select-word"
- "backward-select-word"
- "beginning-of-file"
- "end-of-file"
- "beginning-of-line"
- "end-of-line"
- "select-to-beginning-of-file"
- "select-to-end-of-file"
- "select-to-beginning-of-line"
- "select-to-end-of-line"
- "copy-clipboard"
- "copy-append-clipboard"
- "cut-clipboard"
- "cut-append-clipboard"
- "paste-clipboard"
- "delete-selection"
- "delete-previous-character"
- "delete-next-character"
- "clear-selection"
- "delete-to-end-of-line"
- "delete-next-word"
- "delete-previous-word"
- "delete-line"
- "undo"
- "redo"

See also `add-editor-keymap-functions`.

- (`add-text-keymap-functions` *keymap*) ⇒ void
  keymap : `keymap%` object

`append-editor-font-menu-items`

Appends menu items to a given menu (not a popup menu) to implement a standard set of font-manipulation operations, such as changing the font face or style. The callback for each menu item uses `get-edit-target-object` in `top-level-window<%>` (finding the frame by following a chain of parents until a frame is reached); if the result is an `editor<%>` object, `change-style` in `editor<%>` is called on the editor.

- (`append-editor-font-menu-items` *menu*) ⇒ void
  menu : `menu%` or `popup-menu%` object

`append-editor-operation-menu-items`

Appends menu items to a given menu (not a popup menu) to implement the standard editor operations, such as cut and paste. The callback for each menu item uses `get-edit-target-object` in `top-level-window<%>` (finding the frame by following a chain of parents until a frame is reached); if the result is an `editor<%>` object, `do-edit-operation` in `editor<%>` is called on the editor.

- (`append-editor-operation-menu-items` *menu* *text-only?*) ⇒ void
    menu : `menu%` or `popup-menu%` object
    text-only? = `#t` : boolean

  If *text-only?* is `#f`, then menu items that insert non-text snips (such as Insert Image...) are appended to the menu.

`current-text-keymap-initializer`

Parameter that specifies a keymap-initialization procedure. This procedure is called to initialize the keymap of a `text-field%` object or an `text%` object created by `graphical-read-eval-print-loop`.

The initializer takes a keymap object and returns nothing. The default initializer chains the given keymap to an internal keymap that implements standard text editor keyboard and mouse bindings for cut, copy, paste, undo, and select-all. The right mouse button is mapped to popup an edit menu when the button is released. Under X, start-of-line (Ctl-A) and end-of-line (Ctl-E) are also mapped.

- (`current-text-keymap-initializer`) ⇒ procedure of one argument: a `keymap%` object

  Returns the current initializer procedure.

- (`current-text-keymap-initializer` *proc*) ⇒ void
    proc : procedure of one argument: a `keymap%` object

  Sets the initializer procedure.

`editor-set-x-selection-mode`

- (`editor-set-x-selection-mode` *on*) ⇒ void
    on : boolean

  Under X Windows, editor selections conform to the X Windows selection conventions instead of a clipboard-based convention. If *on* is `#f`, the behavior is switched to the clipboard-based convention (where copy must be explicitly requested before a paste).

`get-the-editor-data-class-list`

- (`get-the-editor-data-class-list`) ⇒ `editor-data-class-list<%>` object

  Gets the editor data class list instance for the current eventspace.

`get-the-snip-class-list`

- (`get-the-snip-class-list`) ⇒ `snip-class-list<%>` object

  Gets the snip class list instance for the current eventspace.

`read-editor-global-footer`

- (`read-editor-global-footer` *in*) ⇒ boolean
    in : `editor-stream-in%` object

  See `read-editor-global-header`. Call `read-editor-global-footer` even if `read-editor-global-header` returns `#f`.

`read-editor-global-header`

- (`read-editor-global-header` *in*) ⇒ boolean
    in : `editor-stream-in%` object

  Reads data from *in* to initialize for reading editors from the stream. The return value is `#t` if the read succeeds, or `#f` otherwise. One or more editors can be read from the stream by calling the editor's `read-from-file` method. (The number of editors to be read must be known by the application beforehand.) When all editors are read, call `read-editor-global-footer`. Calls to `read-editor-global-header` and `read-editor-global-footer` must bracket any call to `read-from-file`, and only one stream at a time can be read using these methods or written using `write-editor-global-header` and `write-editor-global-footer`.

`the-editor-wordbreak-map`

See `editor-wordbreak-map%`.

- `the-editor-wordbreak-map` ⇒ `editor-wordbreak-map%` object
    Initial value : basic wordbreak mapping

`the-style-list`

See `style-list%`.

- `the-style-list` ⇒ `style-list%` object
    Initial value : empty style list

`write-editor-global-footer`

- (`write-editor-global-footer` *out*) ⇒ boolean
    out : `editor-stream-out%` object

  See `write-editor-global-header`. Call `write-editor-global-footer` even if `write-editor-global-header` returns `#f`.

`write-editor-global-header`

- (`write-editor-global-header` *out*) ⇒ boolean
    out : `editor-stream-out%` object

  Writes data to *out* to initialize for writing editors to the stream. The return value is `#t` if the write succeeds, or `#f` otherwise. One or more editors can be written to the stream by calling the editor's `write-to-file` method. When all editors are written, call `write-editor-global-footer`. Calls to `write-editor-global-header` and `write-editor-global-footer` must bracket any call to `write-to-file`, and only one stream at a time can be written using these methods or read using `read-editor-global-header` and `read-editor-global-footer`.

See also section 8.2.

Part IV

Appendices

11.   Running MrEd

MrEd accepts a number of command-line flags. Under MacOS, a user can specify command-line flags by holding down the Command key while starting MrEd, which provides a dialog for entering the command line. Dragging files onto the MrEd icon in MacOS is equivalent to providing each file's name on the command line preceded by `-f`, so each file is loaded after MrEd starts. When files are dragged onto MrEd with the Command key pressed, the command line specified in the dialog is appended to the implicit command-line for loading the files.

MrEd accepts the following flags (in addition to the X-specific flags described in §11.1):

- **Startup file and expression flags:**

  * `-e` *expr* : Evaluates *expr* after MrEd starts.
  * `-f` *file* : Loads *file* after MrEd starts.
  * `-d` *file* : Uses `load/cd` to load *file* after MrEd starts.
  * `-F` : Loads each remaining argument as a file after MrEd starts.
  * `-D` : Loads each remaining argument as a file using `load/cd` after MrEd starts.
  * `-l` *file* : Loads the MzLib library *file* after MrEd starts.
  * `-L` *file collect* : Loads the library *file* in the collection *collect* after MrEd starts.
  * `-r` *file* or `--script` *file* : Use this flag for MrEd-based scripts. It mutes the startup banner printout, suppresses the graphical `read-eval-print` loop, and loads *file* after MrEd starts. No argument after *file* is treated as a flag. The `-r` or `--script` flag is a shorthand for `-fmv-`.
  * `-i` *file* or `--script-cd` *file* : Same as `-r` *file* or `--script` *file*, except that the current directory is changed to *file*'s directory before it is loaded. The `-i` or `--script-cd` flag is a shorthand for `-dmv-`.
  * `-z` or `--stdio` : Calls `read-eval-print` loop (using the current input and output), and suppresses the graphical `read-eval-print` loop. The `-z` or `--stdio` flag is shorthand for `-ve "(read-eval-print-loop)"`.
  * `-w` or `--awk` : Loads the **awk.ss** library after MrEd starts.
  * `-k` *n m* : Loads code embedded in the executable from file position *n* to *m* after MrEd starts. This flag is useful for creating a stand-alone binary by appending code to the normal MrEd executable. See *PLT mzc: MzScheme Compiler Manual* for more details.

- **Initialization flags:**

  * `-x` or `--no-lib-path` : Suppresses the initialization of `current-library-collection-paths` (as described in Library Collections and MzLib, §15 in *PLT MzScheme: Language Manual*).
  * `-q` or `--no-init-file` : Suppresses loading the user's initialization file, as described below.

- **Language setting flags:**

  * `-g` or `--case-sens` : Creates an initial namespace where identifiers and symbols are case-sensitive.
  * `-c` or `--esc-cont` : Creates an initial namespace where `call-with-current-continuation` and `call/cc` capture escape continuations (like `call/ec`) instead of full continuations.
  * `-s` or `--set-undef` : Creates an initial namespace where `set!` will successfully mutate an undefined global variable (implicitly defining it).

* `-a` or `--no-auto-else` : Creates an initial namespace where falling through all of the clauses in a `cond` or `case` expression raises the `exn:else` exception.
* `-n` or `--no-key` : Creates an initial namespace where keywords are not enforced.
* `-y` or `--hash-percent-syntax` : Creates an initial namespace that includes only the `#%` syntactic forms.

- **Miscellaneous flags:**
  * `--` : No argument following this flag is used as a flag.
  * `-m` or `--mute-banner` : Suppresses the startup banner text.
  * `-v` or `--version` : Suppresses the graphical `read-eval-print` loop and prints version information to stdout.
  * `-V` or `--no-yield` : Suppresses the graphical `read-eval-print` loop, prints version information to stdout, and suppresses the normal yield that follows command-line expression evaluation and file loading.
  * `-h` or `--help` : Shows information about MrEd's command-line flags and then exits; ignoring other flags.
  * `-p` or `--persistent` : Catches the SIGDANGER (low page space) signal and ignores it (AIX only).
  * `-R`*file* or `--restore` *file* : Restores a saved image (see Images, §14.8 in *PLT MzScheme: Language Manual*). Extra arguments after *file* are returned as a vector of strings to the continuation of the `write-image-to-file` call that created the image.

Extra arguments following the last flag are put into the Scheme global variable `argv` as a vector of strings. The name used to start MrEd is put into the global variable `program` as a string.

Multiple single-letter flags (the ones preceded by a single dash) can be collapsed into a single flag by concatenating the letters, as long as the the first flag is not `--`. The arguments for each flag are placed after the collapsed flags (in the order of the flags). For example,

   `-vfme` *file expr*

and

   `-v -f` *file* `-m -e` *expr*

are equivalent.

The `current-library-collection-paths` parameter is initialized (as described in Library Collections and MzLib, §15 in *PLT MzScheme: Language Manual*) before any expression or file is evaluated or loaded, unless the `-x` or `--no-lib-path` flag is specified.

Unless the `-q` or `--no-init-file` flag is specified, a user initialization file is loaded after `current-library-collection-paths` parameter is initialized and before any other expression or file is evaluated or loaded. The path to the user initialization file is obtained from MzScheme's `find-system-path` procedure using `'init-file`.

Expressions and files are evaluated and loaded in order that they are provided on the command line. If an error occurs, the remaining expressions and files are skipped. The thread that loads the files and evaluates the expressions is the **main thread**. When the main thread terminates (or is killed), the MrEd process exits. The main thread is also the handler thread of the initial eventspace.

After the command-line files and expressions are loaded and evaluated, the main thread calls `graphical-read-eval-print-loop`, unless the `-v`, `--version`, `-r`, `--script`, `-i`, `--script-cd` flag is specified. The `-z` or `--stdio` flag also suppresses the call to `graphical-read-eval-print-loop`, but

it calls `read-eval-print-loop`, instead. (The other flags, such as `-v`, have no effect on this call to `read-eval-print-loop`.)

Finally, after the command-line files and expressions are loaded and evaluated, and after `graphical-read-eval-print-loop` or `read-eval-print-loop` is called, the main thread evaluates `(yield 'wait)`. The `-V` or `--no-yield` flag suppresses this call to `yield`.

The exit status for the MrEd process indicates an error if an error occurs evaluating or loading a command-line expression or file and `graphical-read-eval-print-loop` is not called afterwards, or if the default exit handler is called with an exact integer between 1 and 255.

Evaluating command-line expressions with `-f` or `-v` is different from evaluating the same expressions within the window provided by `graphical-read-eval-print-loop`. The `graphical-read-eval-print-loop` window creates a new eventspace (and thus a new thread) for evaluating expressions entered into the window. One consequence of this convention is that terminating the evaluation thread (e.g., with `(kill-thread (current-thread))`) does not cause MrEd to exit, because the evaluation thread is not MrEd's main thread.[1]

In contrast, MzScheme's `read-eval-print-loop` always evaluates expressions within the thread that calls `read-eval-print-loop`. Using the `-z` or `--stdio` flag calls `read-eval-print-loop` in the main thread, so `(kill-thread (current-thread))` in that case does exit MrEd. Furthermore, the main thread is the handler thread for the initial eventspace; thus, windows created in `read-eval-print-loop` without changing the eventspace never receive events unless `(yield)` is called explicitly.

11.1   X Window System Flags

Under Unix/X, the following standard X Window System flags are recognized (but not necessarily implemented): `-display` (1 argument), `-geometry` (1 argument), `-bg` (1 argument), `-background` (1 argument), `-fg` (1 argument), `-foreground` (1 argument), `-fn` (1 argument), `-font` (1 argument), `-iconic` (0 arguments), `-name` (1 argument), `-rv` (0 argument), `-reverse` (0 arguments), `-rv` (0 arguments), `-selectionTimeout` (1 argument), `-synchronous` (0 arguments), `-title` (1 argument), `-xnllanguage` (1 argument), and `-xrm` (1 argument).

All X flags must precede all other flags and arguments.

11.2   Initial Eventspace

MrEd creates an initial eventspace with a parameterization obtained from the parameterizaton branch handler in the initial parameterization. The handler thread for this eventspace is MrEd's main thread; if this thread is killed, then the MrEd process exits.

---

[1]However, the exit handler is not changed, so evaluating `(exit)` does exit MrEd unless the exit handler is changed before calling `graphical-read-eval-print-loop`.

Index