PLT Framework: GUI Application Framework

Robert Bruce Findler (robby@plt-scheme.org)
Matthew Flatt (mflatt@plt-scheme.org)

206
Released January 2004

Copyright notice

Contents

1. This Manual

This manual describes a framework for programmers developing applications MrEd. It assumes familiarity with MrEd as described in *PLT MrEd: Graphical Toolbox Manual* and MzScheme as described in *PLT MzScheme: Language Manual*.

[build date: January 27, 2004]

1.1 Thanks

Thanks to Shriram Krishnamurthi, Cormac Flanagan, Matthias Felleisen, Ian Barland, Gann Bierner, Richard Cobbe, Dan Grossman, Stephanie Weirich, Paul Steckler, Sebastian Good, Johnathan Franklin, Mark Krentel, Corky Cartwright, Michael Ernst, Kennis Koldewyn, Bruce Duba, and many others for their feedback and help.

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

This manual was typeset on January 27, 2004.

2. Overview

The Framework is a library that provides application framework for MrEd. It is designed to make implementation of an application in MrEd simpler and easier. It provides standard classes and utilities for managing

- frames, §**??**,
- editors, §**??**,
- and many others.

See section 3.1 for information on how to load the framework into an application.

3. Preliminaries

3.1 Libraries

The framework provides these libraries:

- **Entire Framework**
  - (**require** (**lib** "framework.ss" "framework"))
    This library provides all of the definitions and syntax described in this manual.
  - (**require** (**lib** "framework-sig.ss" "framework"))
    This library provides the signature definitions: #1^, and #1^. The *framework^* signature cotains all of the names of the procedures described in this manual, except those that begin with *test:* and *gui-utils:*. The *framework-class^* signature contains all of the classes defined in this manual.
  - (**require** (**lib** "framework-unit.ss" "framework"))
    This library provides one *unit/sigs*, § in *PLT MzLib: Libraries Manual*: #1@. It exports the signature *framework^*. It imports the *mred^* signature.
  - (**require** (**lib** "macro.ss" "framework"))
    This defines the mixin macro. See §3.2 for more information.

- **Test Suite Engine**

  (**require** (**lib** "test.ss" "framework"))

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

- **GUI Utilities** (**require** (**lib** "gui-utils.ss" "framework"))

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

3.2 Mixins

The framework relies heavily on mixins. A mixin is a class parameterization modeled on a paper published by Flatt, Felleisen, and Krishnamurthi, available at `http://www.cs.utah.edu/plt/publications/icfp98-ff/`. The implementation of these mixins in MzScheme is with the combination of **lambda** and **class**. The framework provides a macro to simplify the checking and implementation of these mixins. Its syntax is very similar to the syntax for `class*`, § in *PLT MzLib: Libraries Manual*. The shape of a mixin is:

(**mixin** (*interface-expr* ...) (*interface-expr* ...)
  instance-variable-clause ...)

This macro expands into a procedure that accepts a class. The argument passed to this procedure must match the interfaces of the first *interface-expr*s expressions. The procedure returns a class that is derived from its argument. This result class must match the interfaces specified in the second *interface-expr*s section; it has clauses specified by *instance-variable-clause*s. The syntax of the *initialization-variables* and *instance-variable-clause* are exactly the same as `class*/names`, § in *PLT MzLib: Libraries Manual*.

The **mixin** macro does some checking to be sure that variables that the *instance-variable-clause*s refer to in their super class are in the interfaces. That checking and the checking that the input class matches the declared interfaces aside, the mixin macro's expansion is something like this:

```
(mixin (i<%> ...) (j<%> ...)
  clause ...)
=
(lambda (%)
  (class* % (j<%> ...)
    clause ...))
```

The *i*<%> interfaces do not appear in the output because they are only used for the error checking and are discarded by the time the class is created.

The **mixin** macro is provided by

```
(require (lib "macro.ss" "framework"))
```

3.3 GUI Test Suite Utilities

The framework provides several new primitive functions that simulate user actions, which may be used to test applications. You use these primitives and combine them just as regular MzScheme functions. For example,

```
(begin
  (test:keystroke #\A)
  (test:menu-select "File" "Save"))
```

sends a keystroke event to the window with the keyboard focus and invokes the callback function for the "Save" menu item from the "File" menu. This has the same effect as if the user typed the key "A", pulled down the "File" menu and selected "Save".

It is possible to load this portion of the framework without loading the rest of the framework. See fw:libraries for more details.

Currently, the test engine has primitives for pushing buttons, setting check-boxes and choices, sending keystrokes, selecting menu items and clicking the mouse. Many functions that are also useful in application testing, such as traversing a tree of panels, getting the text from a canvas, determining if a window is shown, and so on, exist in MrEd.

3.3.1 Actions and completeness

The actions associated with a testing primitive may not have finished when the primitive returns to its caller. Some actions may yield control before they can complete. For example, selecting "Save As..." from the "File" menu opens a dialog box and will not complete until the "OK" or "Cancel" button is pushed.

However, all testing functions wait at least a minimum interval before returning to give the action a chance to finish. This interval controls the speed at which the test suite runs, and gives some slack time for events to complete. The default interval is 100 milliseconds. The interval can be queried or set with `test:run-interval`.

A primitive action will not return until the run-interval has expired and the action has finished, raised an error, or yielded. The number of incomplete actions is given by `test:number-pending-actions`.

Note: Once a primitive action is started, it is not possible to undo it or kill its remaining effect. Thus, it is not possible to write a utility that flushes the incomplete actions and resets number-pending-actions to zero.

However, actions which do not complete right away often provide a way to cancel themselves. For example, many dialog boxes have a "Cancel" button which will terminate the action with no further effect. But this is accomplished by sending an additional action (the button push), not by undoing the original action.

3.3.2 Errors

Errors in the primitive actions (which necessarily run in the handler thread) are caught and reraised in the calling thread.

However, the primitive actions can only guarantee that the action has started, and they may return before the action has completed. As a consequence, an action may raise an error long after the function that started it has returned. In this case, the error is saved and reraised at the first opportunity (the next primitive action).

The test engine keeps a buffer for one error, saving only the first error. Any subsequent errors are discarded. Reraising an error empties the buffer, allowing the next error to be saved.

The function `test:reraise-error` reraises any pending errors.

3.3.3 Technical Issues

Active Frame

The Self Test primitive actions all implicitly apply to the top-most (active) frame.

Thread Issues

The code started by the primitive actions must run in the handler thread of the eventspace where the event takes place. As a result, the test suite that invokes the primitive actions must *not* run in that handler thread (or else some actions will deadlock). See the eventspace section, §2.4 in *PLT MrEd: Graphical Toolbox Manual* for more info.

Window Manager (Unix only)

In order for the Self Tester to work correctly, the window manager must set the keyboard focus to follow the active frame. This is the default behavior in Microsoft Windows and MacOS, but not in X windows.

In X windows, you must explicitly tell your window manager to set the keyboard focus to the top-most frame, regardless of the position of the actual mouse. Some window managers may not implement such functionality. You can obtain such an effect in Fvwm and Fvwm95 by using the option:

```
Style  "*"  ClickToFocus
```

4. Application

The procedure in this chapter is used to supply information about your application to the framework.

5. Autosave

Autosaving in MrEd is performed by registering with a single autosaver daemon. Objects that are registered with the autosaver must have a `do-autosave` method that is called periodically with no arguments.

- `autosave:autosavable<%>`

5.1 `autosave:autosavable<%>`

Classes that implement this interface can be autosaved.

`do-autosave`

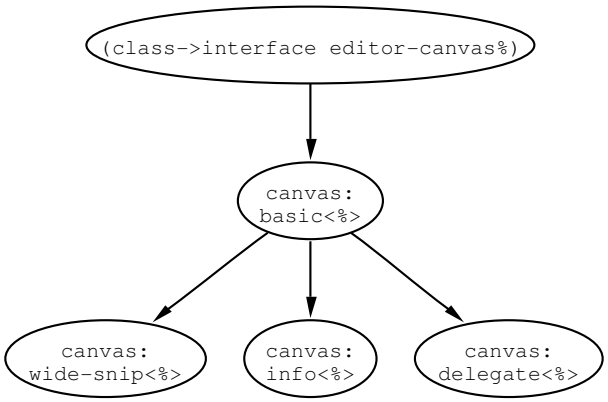
This method is called when the object is registered to be autosaved (see `autosave:register`).

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

6. Canvas

This chapter describes the editor canvas mixins, interfaces, and classes.

This is the interface hierarchy:



- `canvas:basic%`
- `canvas:info%`
- `canvas:wide-snip%`
- `canvas:basic<%>`
- `canvas:delegate<%>`
- `canvas:info<%>`
- `canvas:wide-snip<%>`
- `canvas:basic-mixin`
- `canvas:delegate-mixin`
- `canvas:info-mixin`
- `canvas:wide-snip-mixin`

6.1 `canvas:basic<%>`

Extends: `(class->interface editor-canvas%)`

6.2 canvas:basic-mixin

Domain: (class->interface editor-canvas%)

Implements: canvas:basic<%>

- **(new canvas:basic-mixin% (parent _) [(editor _)] [(style _)] [(scrolls-per-page _)] [(label _)] [(wheel-step _)] [(line-count _)] [(horizontal-inset _)] [(vertical-inset _)] [(enabled _)] [(vert-margin _)] [(horiz-margin _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)]) ⇒ canvas:basic-mixin% 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 control-border deleted)
  scrolls-per-page = 100 : exact integer in [1, 10000]
  label = #f : string (up to 200 characters) or #f
  wheel-step = 3 : exact integer in [1, 10000] or #f
  line-count = #f : exact integer in [1, 1000] or #f
  horizontal-inset = 5 : exact integer in [1, 1000]
  vertical-inset = 5 : exact integer in [1, 1000]
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

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

  The *style* list can contain the following flags:

  – 'no-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
  – 'control-border — gives the canvas a border that is like a text-field% control
  – 'deleted — creates the canvas as initially hidden and without affecting *parent*'s geometry; the canvas can be made active later by calling *parent*'s add-child method

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

  If provided, the *wheel-step* argument is passed on to the wheel-step method. The default wheel step can be overridden globally though the '|MrEd:wheelStep| preference; see "Preferences" (§12 in *PLT MrEd: Graphical Toolbox Manual*).

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

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

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

6.3 `canvas:delegate<%>`

Extends: `canvas:basic<%>`

This class is part of the delegate window implementation.

6.4 `canvas:delegate-mixin`

Domain: `canvas:basic<%>`

Implements: `canvas:delegate<%>`

Implements: `canvas:basic<%>`

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

`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 canceled.

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

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

6.5 `canvas:info<%>`

Extends: `canvas:basic<%>`

6.6 `canvas:info-mixin`

Domain: `canvas:basic<%>`

Implements: `canvas:basic<%>`

Implements: `canvas:info<%>`

`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-canvas:info-mixin* `on-focus`**)** ⇒ void

  Enables or disables the caret in the display's editor, if there is one.

  sets the canvas that the frame displays info about.

`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** *a-canvas:info-mixin* `set-editor`**)** ⇒ void

  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 minimum height is adjusted.

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

6.7 canvas:wide-snip<%>

Extends: `canvas:basic<%>`

Any `canvas%` that matches this interface will automatically resize selected snips when it's size changes. Use `add-tall-snip` and `add-wide-snip` to specify which snips should be resized.

`add-tall-snip`

Snips passed to this method will be resized when the canvas's size changes. Their height will be set so they take up all of the space from their tops to the bottom of the canvas.

- **(send** *a-canvas:wide-snip* `add-tall-snip` *snip***)** ⇒ void
  snip : (instance `snip%`)

`add-wide-snip`

Snips passed to this method will be resized when the canvas's size changes. Their width will be set so they take up all of the space from their lefts to the right edge of the canvas.

- **(send** *a-canvas:wide-snip* `add-wide-snip` *snip***)** ⇒ void
  snip : (instance `snip%`)

`recalc-snips`

Recalculates the sizes of the wide snips.

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

6.8 canvas:wide-snip-mixin

Domain: canvas:basic<%>

Implements: canvas:basic<%>

Implements: canvas:wide-snip<%>

This canvas mantains a list of wide and tall snips and adjusts their heights and widths when the canvas's size changes.

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

`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-canvas:wide-snip-mixin* `on-size` *width height***)** ⇒ void
  width : exact integer in [0, 10000]
  height : exact integer in [0, 10000]

  If the canvas is displaying an editor, its on-display-size method is called.

  Adjusts the sizes of the marked snips.

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

6.9 canvas:basic% = (canvas:basic-mixin editor-canvas%)

```
canvas:basic% = (canvas:basic-mixin editor-canvas%)
```

- **(new** `canvas:basic%` **(parent** _**)** **[(editor** _**)]** **[(style** _**)]** **[(scrolls-per-page** _**)]** **[(label** _**)]** **[(wheel-step** _**)]** **[(line-count** _**)]** **[(horizontal-inset** _**)]** **[(vertical-inset** _**)]** **[(enabled** _**)]** **[(vert-margin** _**)]** **[(horiz-margin** _**)]** **[(min-width** _**)]** **[(min-height** _**)]** **[(stretchable-width** _**)]** **[(stretchable-height** _**)])** ⇒ canvas:basic% 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 control-border deleted)`
  scrolls-per-page = 100 : exact integer in [1, 10000]
  label = #f : string (up to 200 characters) or #f
  wheel-step = 3 : exact integer in [1, 10000] or #f
  line-count = #f : exact integer in [1, 1000] or #f

horizontal-inset = 5 : exact integer in [1, 1000]
vertical-inset = 5 : exact integer in [1, 1000]
enabled = #t : boolean
vert-margin = 0 : exact integer in [0, 1000]
horiz-margin = 0 : exact integer in [0, 1000]
min-width = 0 : exact integer in [0, 10000]
min-height = 0 : exact integer in [0, 10000]
stretchable-width = #t : boolean
stretchable-height = #t : boolean

Passes all arguments to `super-init`.

6.10 canvas:info% = (canvas:info-mixin canvas:basic%)

`canvas:info% = (canvas:info-mixin canvas:basic%)`

- **(new canvas:info% (parent _) [(editor _)] [(style _)] [(scrolls-per-page _)] [(label _)] [(wheel-step _)] [(line-count _)] [(horizontal-inset _)] [(vertical-inset _)] [(enabled _)] [(vert-margin _)] [(horiz-margin _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)]) ⇒** canvas:info% 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 control-border deleted)
  scrolls-per-page = 100 : exact integer in [1, 10000]
  label = #f : string (up to 200 characters) or #f
  wheel-step = 3 : exact integer in [1, 10000] or #f
  line-count = #f : exact integer in [1, 1000] or #f
  horizontal-inset = 5 : exact integer in [1, 1000]
  vertical-inset = 5 : exact integer in [1, 1000]
  enabled = #t : boolean
  vert-margin = 0 : exact integer in [0, 1000]
  horiz-margin = 0 : exact integer in [0, 1000]
  min-width = 0 : exact integer in [0, 10000]
  min-height = 0 : exact integer in [0, 10000]
  stretchable-width = #t : boolean
  stretchable-height = #t : boolean

  Passes all arguments to `super-init`.

6.11 canvas:wide-snip% = (canvas:wide-snip-mixin canvas:basic%)

`canvas:wide-snip% = (canvas:wide-snip-mixin canvas:basic%)`

- **(new canvas:wide-snip% (parent _) [(editor _)] [(style _)] [(scrolls-per-page _)] [(label _)] [(wheel-step _)] [(line-count _)] [(horizontal-inset _)] [(vertical-inset _)] [(enabled _)] [(vert-margin _)] [(horiz-margin _)] [(min-width _)] [(min-height _)] [(stretchable-width _)] [(stretchable-height _)]) ⇒** canvas:wide-snip% 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 control-border deleted)
  scrolls-per-page = 100 : exact integer in [1, 10000]

label = #f : string (up to 200 characters) or #f
wheel-step = 3 : exact integer in [1, 10000] or #f
line-count = #f : exact integer in [1, 1000] or #f
horizontal-inset = 5 : exact integer in [1, 1000]
vertical-inset = 5 : exact integer in [1, 1000]
enabled = #t : boolean
vert-margin = 0 : exact integer in [0, 1000]
horiz-margin = 0 : exact integer in [0, 1000]
min-width = 0 : exact integer in [0, 10000]
min-height = 0 : exact integer in [0, 10000]
stretchable-width = #t : boolean
stretchable-height = #t : boolean

Passes all arguments to `super-init`.

7. Color

- color:text-mode%
- color:text%
- color:text-mode<%>
- color:text<%>
- color:text-mixin
- color:text-mode-mixin

7.1 color:text<%>

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

backward-containing-sexp

- (**send** *a-color:text* backward-containing-sexp) ⇒ void
- (**send** *a-color:text* backward-containing-sexp *position cutoff*) ⇒ (union natural-number? false?)
  position : natural-number?
  cutoff : natural-number?

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

  Must only be called while the tokenizer is started.

backward-match

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

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

  Must only be called while the tokenizer is started.

classify-position

- (**send** *a-color:text* classify-position *position*) ⇒ symbol
  position : natural-number?

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

Must only be called while the tokenizer is started.

force-stop-colorer

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

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

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

forward-match

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

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

  Must only be called while the tokenizer is started.

freeze-colorer

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

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

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

insert-close-paren

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

  Position is the place to put the parenthesis and char is the parenthesis to be added. If fixup? is true, the right kind of closing parenthesis will be chosen from the pairs list kept last passed to start-colorer, otherwise char will be inserted, even if it is not the right kind. If flash? is true the matching open parenthesis will be flashed.

is-frozen?

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

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