PLT Miscellaneous Libraries: Reference Manual

PLT (scheme@plt-scheme.org)

206

Released January 2004

Copyright notice

Copyright ©1996-2003 PLT

Permission to make digital/hard copies and/or distribute this documentation for any purpose is hereby granted without fee, provided that the above copyright notice, author, and this permission notice appear in all copies of this documentation.

Send us your Web links

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

Contents

1	Mis	cellaneous Libraries	T
2	Viev	wport Graphics	2
	2.1	Basic Commands	2
	2.2	Position Operations	2
	2.3	Color Operations	3
	2.4	Draw, Clear and Flip Operations	3
		2.4.1 Viewports	4
		2.4.2 Pixels	4
		2.4.3 Lines	4
		2.4.4 Rectangles	4
		2.4.5 Ellipses	5
		2.4.6 Polygons	6
		2.4.7 Strings	6
		2.4.8 Pixmaps	6
	2.5	Miscellaneous Operations	7
	2.6	An Example	7
	2.7	A More Complicated Example	7
	2.8	Protecting Graphics Operations	8
	2.9	Mouse Operations	8
	2.10	Keyboard Operations	9
	2.11	Unitized Graphics	9
3	Turt	tles	11
J			
	3.1	Turtles	11

CONTI	ENTS	CONTENTS		
3.2	Value Turtles			
Index		15		

1. Miscellaneous Libraries

This manual documents miscellaneous libraries distributed with DrScheme.

2. Viewport Graphics

The viewport graphics library is a relatively simple toolbox of graphics commands. The library is not very powerful; it is intended as a simplified alternative to MrEd's full graphical toolbox.

The graphics library originated as SIXlib, a library of X Windows commands available within Chez Scheme at Rice University. The functionality of that library has been reproduced (with backward compatibility) in this version.

To use the viewport graphics library in a PLT language, load it via require:

```
(require (lib "graphics.ss" "graphics"))
```

which loads the **graphics.ss** library from the **graphics** collection. All of the names defined in this chapter will then be available.

2.1 Basic Commands

- (open-graphics)
 Initializes the library's graphics routines. It must be called before any other graphics operations.
- (close-graphics)

 Closes all of the windows and until open-graphics is called again, no graphics routines will work.
- (open-viewport name horiz vert)

 Takes a string name and integers horiz and vert and creates a new window called name. The window is horiz pixels wide and vert pixels high. For backward compatibility, a single posn value (see below) can be submitted in the place of horiz and vert. open-viewport returns a viewport descriptor.
- (open-pixmap name horiz vert)

 Like open-viewport, but the resulting viewport is not displayed on the screen. Offscreen pixmaps are useful for executing a sequence of drawing commands and displaying them all at once with copy-viewport.

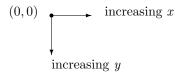
Offscreen pixmaps are also useful in conjunction with viewport-¿snip (see below). This allows functions to compute with graphical objects and view the graphics when results are returned to the interactions window.

• (close-viewport *viewport*)

Takes a viewport descriptor. It removes the viewport from the screen and makes subsequent operations dealing with the viewport illegal.

2.2 Position Operations

A position is a pixel location within a viewport. The upper-left corner is pixel (0,0) and the orientation of the position coordinates within a viewport is as follows:



- (make-posn x y)
 Takes two integers and returns a position with the specified x and y coordinates.
- (posn-x p), (posn-y p)
 Return the x and y coordinates, respectively, of a position.
- (posn? v)
 Reports whether v is a position.
- ((get-pixel viewport) p)
 Returns the color of the pixel at position p in viewport; 0 denotes white and 1 denotes not white.
- ((get-color-pixel *viewport*) p)
 Returns an RGB value for color of the pixel at position p in *viewport*.
- ((test-pixel *viewport*) *color*)
 Returns the color that will actually be used if *color* is used to draw.

2.3 Color Operations

A color can be represented in three ways: as a color index (an integer in 0 to 299, inclusive), as a color name string, or as a rgb value. All drawing functions which take a color argument accept colors in any form. An rgb value is assigned to an index with change-color.

- (make-rgb red green blue)
 Takes three values in the range 0 (dark) to 1 (bright) and returns an rgb (a color).
- (rgb-red color)
 (rgb-blue color)
 (rgb-green color)
 Return the red, green, and blue components, respectively, of a color.
- (rgb? v)
 Reports whether v is a color.
- (change-color index rgb)
 Changes the color at index in the color table to the color specified in rgb. Only the first twenty-one indices are initialized; a color index should not be used until it has been initialized.
- (default-display-is-color?)
 Returns #t if the default display screen for viewports is in color or #f otherwise.

2.4 Draw, Clear and Flip Operations

These are the basic graphics operations for drawing to a viewport. Each function takes a viewport as its argument and returns a function operating within that viewport. Further arguments, if any, are curried. For example, (draw-line viewport) returns a function, that can then be applied to the proper arguments to draw a line in the viewport corresponding to viewport descriptor viewport. An example follows.

Where "draw-" commands make pixels black, "clear-" commands make them white.

Where "draw-" commands make pixels black, a "flip-" commands cause them to change.

2.4.1 Viewports

• ((draw-viewport *viewport*) *color*)

Takes a viewport descriptor. It returns a function that colors the entire contents of *viewport*. The optional *color* argument defaults to black.

• ((clear-viewport *viewport*))

Takes a viewport descriptor. It returns a function that whitens the entire contents of viewport.

• ((flip-viewport *viewport*))

Takes a viewport descriptor. It returns a function that flips the contents of viewport.

• (copy-viewport source-viewport destination-viewport)

Takes two viewport descriptors. It copies the source-viewport into the destination-viewport.

2.4.2 Pixels

• ((draw-pixel viewport) p color)

Takes a viewport descriptor. It returns a function that draws a pixel in *viewport* at the specified position. The optional *color* argument defaults to black.

• ((clear-pixel *viewport*) p)

Takes a viewport descriptor. It returns a function that clears a pixel in *viewport* at the specified position.

• ((flip-pixel viewport) p)

Takes a viewport descriptor. It returns a function that flips a pixel in *viewport* at the specified position.

2.4.3 Lines

• ((draw-line viewport) p1 p2 color)

Takes a viewport descriptor. It returns a function that draws a line in the *viewport* connecting positions p1 and p2. The optional *color* argument defaults to black.

• ((clear-line viewport) p1 p2)

Takes a viewport descriptor. It returns a function that clears a line in *viewport* connecting positions p1 and p2.

• ((flip-line viewport) p1 p2)

Takes a viewport descriptor. It returns a function that flips a line in *viewport* connecting positions p1 and p2.

2.4.4 Rectangles

• ((draw-rectangle viewport) posn width height color)

Takes a viewport descriptor. It returns a function that draws a rectangle border in the *viewport* with the top-left of the rectangle at the position *posn* and with sides *width* across and *height* tall. The optional *color* argument defaults to black.

• ((clear-rectangle viewport) posn width height)

Takes a viewport descriptor. It returns a function that clears a rectangle border in the viewport with

the top-left of the rectangle at the position posn and with sides width across and height tall. The optional color argument defaults to black.

- ((flip-rectangle viewport) posn width height color)
 - Takes a viewport descriptor. It returns a function that flips a rectangle border in the *viewport* with the top-left of the rectangle at the position *posn* and with sides *width* across and *height* tall. The optional *color* argument defaults to black.
- ((draw-solid-rectangle viewport) posn width height color)

Takes a viewport descriptor. It returns a function that paints a solid rectangle in the *viewport* with the top-left of the rectangle at the position *posn* and with sides *width* across and *height* tall. The optional *color* argument defaults to black.

- ((clear-solid-rectangle viewport) posn width height)
 - Takes a viewport descriptor. It returns a function that erases a solid rectangle in the *viewport* with the top-left of the rectangle at the position *posn* and with sides *width* across and *height* tall. The optional *color* argument defaults to black.
- ((flip-solid-rectangle viewport) posn width height color)

 Takes a viewport descriptor. It returns a function that flips a solid rectangle in the viewport with the top-left of the rectangle at the position posn and with sides width across and height tall. The optional color argument defaults to black.

2.4.5 Ellipses

- ((draw-ellipse viewport) posn width height color)
 - Takes a viewport descriptor. It returns a function that draws an ellipse border in the *viewport*. The *posn*, *width*, and *height* arguments are as in draw-rectangle; the ellipse is inscribed within the specified rectangle. The optional *color* argument defaults to black.
- ((clear-ellipse viewport) posn width height)

Takes a viewport descriptor. It returns a function that clears an ellipse border in the *viewport*. The *posn*, *width*, and *height* arguments are as in clear-rectangle; the ellipse is inscribed within the specified rectangle. The optional *color* argument defaults to black.

- ((flip-ellipse viewport) posn width height color)
 - Takes a viewport descriptor. It returns a function that flips an ellipse border in the *viewport*. The *posn*, width, and height arguments are as in flip-rectangle; the ellipse is inscribed within the specified rectangle. The optional *color* argument defaults to black.
- ((draw-solid-ellipse viewport) posn width height color)

Takes a viewport descriptor. It returns a function that paints a solid ellipse in the *viewport*. The *posn*, width, and height arguments are as in draw-rectangle; the ellipse is inscribed within the specified rectangle. The optional color argument defaults to black.

- ((clear-solid-ellipse viewport) posn width height)
 - Takes a viewport descriptor. It returns a function that erases a solid ellipse in the *viewport*. The *posn*, *width*, and *height* arguments are as in clear-rectangle; the ellipse is be inscribed within the specified rectangle. The optional *color* argument defaults to black.
- ((flip-solid-ellipse viewport) posn width height color)
 - Takes a viewport descriptor. It returns a function that flips a solid ellipse in the *viewport*. The *posn*, *width*, and *height* arguments are as in flip-rectangle; the ellipse is be inscribed within the specified rectangle. The optional *color* argument defaults to black.

2.4.6 Polygons

• ((draw-polygon viewport) posn-list posn color)

Takes a viewport descriptor. It returns a function that draws a polygon border in the *viewport* using *posn-list* for the polygon vertices and *posn* as an offset for the polygon. The optional *color* argument defaults to black.

• ((clear-polygon *viewport*) *posn-list posn*)

Takes a viewport descriptor. It returns a function that erases a polygon border in the *viewport* using *posn-list* for the polygon vertices and *posn* as an offset for the polygon.

• ((flip-polygon viewport) posn-list posn)

Takes a viewport descriptor. It returns a function that flips a polygon border in the *viewport* using *posn-list* for the polygon vertices and *posn* as an offset for the polygon.

• ((draw-solid-polygon viewport) posn-list posn color)

Takes a viewport descriptor. It returns a function that paints a solid polygon in the *viewport* using *posn-list* for the polygon vertices and *posn* as an offset for the polygon. The optional *color* argument defaults to black.

• ((clear-solid-polygon *viewport*) *posn-list posn*)

Takes a viewport descriptor. It returns a function that erases a solid polygon in the *viewport* using *posn-list* for the polygon vertices and *posn* as an offset for the polygon.

• ((flip-solid-polygon viewport) posn-list posn)

Takes a viewport descriptor. It returns a function that flips a solid polygon in the *viewport* using *posn-list* for the polygon vertices and *posn* as an offset for the polygon.

2.4.7 Strings

• ((draw-string *viewport*) p string color)

Takes a viewport descriptor. It returns a function that draws a string at a specified location in the viewport. The lower left of the string begins at p. The optional color argument defaults to black.

• ((clear-string *viewport*) p string)

Takes a viewport descriptor. It returns a function that clears a string at a specified location in *viewport*. The lower left of the string begins at p.

• ((flip-string *viewport*) p string)

Takes a viewport descriptor. It returns a function that flips a string at a specified location in *viewport*. The lower left of the string begins at p.

2.4.8 Pixmaps

• (((draw-pixmap-posn filename type) viewport) posn color)

Draws a pixmap into *viewport* with its upper left corner at position *posn*. The *type* is an optional symbol, one of 'gif, 'gif/mask, 'xbm, 'xpm, 'bmp, 'pict, 'unknown, or 'unknown/mask, and defaults to 'unknown/mask. If *type* is 'unknown or 'unknown/mask, then the content of the file is examined to determine the type. All formats are supported on all platforms, except 'pict which is only supported under Mac OS. The 'gif/mask and 'unknown/mask types draw the bitmap with a transparent background if *filename* refers to a GIF file with a transparent background.

The argument color is only used when the pixmap is black and white. In that case, the color is used instead of black in the drawn image.

- ((draw-pixmap viewport) filename p color)

 Draws a pixmap into viewport w with its upper left corner at position p. If color is not #f it is passed to set-viewport-pen with the viewport. It defaults to #f.
- ((save-pixmap viewport) filename type)
 Saves the current content of viewport to filename. The type is an optional symbol, one of 'xbm, 'xpm, 'bmp (Windows only), or 'pict (Mac OS only); the default is 'xpm.

2.5 Miscellaneous Operations

• ((get-string-size *viewport*) *string*)

Takes a viewport descriptor. It returns a function that returns the size of a string as a list of two numbers: the width and height.

• (viewport->snip viewport)

Takes a viewport descriptor. It returns an object that can be inserted into an editor buffer to display the current image in the viewport. (Subsequent drawing to the viewport does not affect the snip's image.)

When snips are the results of computations in the interactions window, DrScheme will print show the contents of the viewport, right in the interactions window.

• (viewport-dc *viewport*)

Takes a viewport descriptor. It returns an object that can be used with the primitive MrEd toolbox functions to draw into the viewport's on-screen representation (if any). Mirror all such drawing to the result of (viewport-offscreen-dc viewport), too.

• (viewport-offscreen-dc *viewport*)

Takes a viewport descriptor. It returns an object that can be used with the primitive MrEd toolbox functions to draw into the viewport's off-screen representation. Mirror all such drawing to the result of (viewport-dc viewport), too.

2.6 An Example

```
(open-graphics)
;; nothing appears to happen, but the library is initialized...
(define w (open-viewport "practice" 300 300))
;; viewport appears
((draw-line w) (make-posn 30 30) (make-posn 100 100))
;; line appears
(close-viewport w)
;; viewport disappears
(close-graphics)
;; again, nothing appears to happen, but
;; unclosed viewports (if any) would disappear
```

2.7 A More Complicated Example

The use of multiple viewports, viewport descriptors, drawing operations for multiple viewports is as easy as the use of a single viewport:

2.8 Protecting Graphics Operations

To guarantee the proper closing of viewports in cases of errors, especially when a program manages several viewports simultaneously, a programmer should use dynamic-wind:

```
(let ([w (open-viewport "hello" 100 100)])
  (dynamic-wind
    ;; what we want to happen first: nothing
    void
    ;; the main program (errors constrained to this piece)
    (lambda () (draw-pixel 13)) ; an error
    ;; what we would like to happen, whether the main program finishes
    ;; normally or not
    (lambda () (close-viewport w))))
```

2.9 Mouse Operations

The graphics library contains functions that determine where the mouse is, if there are any clicks, etc. The functions get-mouse-click and ready-mouse-click first return a "mouse-click descriptor," and then other functions take the descriptor and return the mouse's position, which button was pushed, etc. Mouse clicks are buffered and returned in the same order in which they occurred. Thus, the descriptors returned by get-mouse-click and ready-mouse-click may be from clicks that occurred long before these functions were called.

- (get-mouse-click viewport)

 Takes a viewport descriptor and returns a mouse click descriptor. It returns the next mouse click in the viewport, waiting for a click if necessary.
- (ready-mouse-click viewport)

 Takes a viewport descriptor and returns either a mouse click descriptor, or else #f if none is available.

 Unlike the previous function, ready-mouse-click returns immediately.
- (ready-mouse-release *viewport*)

 Takes a viewport descriptor and returns either a click descriptor from a mouse-release (button-up) event, or else #f if none is available.

• (query-mouse-posn *viewport*)

Takes a viewport descriptor and returns either the position of the mouse cursor within the *viewport*, or else #f if the cursor is currently outside the *viewport*.

• (mouse-click-posn mouse-click)

Takes a mouse click descriptor and returns the position of the pixel where the click occurred.

• (left-mouse-click? *mouse-click*)

Takes a mouse click descriptor and returns #t if the click occurred with the left mouse button, or else #f.

$\bullet \ \ (\texttt{middle-mouse-click?} \ \ mouse\text{-}click) \\$

Similar to left-mouse-click?.

ullet (right-mouse-click? mouse-click)

Similar to left-mouse-click?.

• (viewport-flush-input *viewport*)

As noted above, mouse clicks are buffered. viewport-flush-input takes a viewport descriptor and empties the input buffer of mouse and keyboard events. This action is useful in some real-time applications.

2.10 Keyboard Operations

The graphics library contains functions that report key presses from the keyboard. The functions get-key-press and ready-key-press return a "key-press descriptor," and then key-value takes the descriptor and returns a character or symbol (usually a character) representing the key that was pressed. Key presses are buffered and returned in the same order in which they occurred. Thus, the descriptors returned by get-key-press and ready-key-press may be from presses that occurred long before these functions were called.

• (get-key-press *viewport*)

Takes a viewport descriptor and returns a key press descriptor. It returns the next key press in the *viewport*, waiting for a click if necessary.

• (ready-key-press *viewport*)

Takes a viewport descriptor and returns either a key press descriptor, or else #f if none is available. Unlike the previous function, ready-key-press returns immediately.

• (key-value *key-press*)

Takes a key press descriptor and returns a character or special symbol for the key that was pressed. For example, the Enter key generates #\return, and the up-arrow key generates 'up. For a complete list of possible return values, see *PLT MrEd: Graphical Toolbox Manual*.

• (viewport-flush-input viewport)

As noted above, key presses are buffered. viewport-flush-input takes a viewport descriptor and empties the input buffer of mouse and keyboard events. This action is useful in some real-time applications.

2.11 Unitized Graphics

To use a unitized version of the graphics library (see *PLT MzLib*: *Libraries Manual* for more information on units), get the signatures graphics, graphics:posn-less, and graphics:posn- with:

```
(require (lib "graphics-sig.ss" "graphics"))
```

The graphics signature includes all of the names defined in this chapter. The graphics:posn-less signature contains everything except the posn structure information, and graphics:posn contains only the posn structure.

To obtain graphics@, which imports mred^ (all of the MrEd classes, functions, and constants) and exports graphics^:

```
(require (lib "graphics-unit.ss" "graphics"))
```

The **graphics-posn-less-unit.ss** library provides **graphics-posn-less@**, which imports **graphics:posn^** in addition to MrEd.

3. Turtles

3.1 Turtles

There are two ways to use the turtles in DrScheme. You can use it as a TeachPack (see the DrScheme manual for details of TeachPacks) or as a library. Use the **turtles.ss** TeachPack.

In the MrEd language or in a module, load turtles with

```
(require (lib "turtles.ss" "graphics"))
```

The following are the turtle functions:

- (turtles b) shows and hides the turtles window based on the boolean b. The parameter b is optional; if it is left out, it toggles the state of the turtles.
- (move n) moves the turtle n pixels.
- (draw n) moves the turtle n pixels and draws a line on that path.
- (erase n) moves the turtle n pixels and erases along that path.
- (move-offset h v), (draw-offset h v), (erase-offset h v) are just like move, draw and erase, except they take a horizontal and vertical offset from the turtle's current position.
- (turn theta) turns the turtle theta degrees counter-clockwise.
- (turn/radians theta) turns the turtle theta radians counter-clockwise.
- (clear) erases the turtles window.

Turtles also defines these syntactic forms:

• (split E) spawns a new turtle where the turtle is currently located. In order to distinguish the two turtles, only the new one evaluates the expression E. For example, if you start with a fresh turtle-window and type:

you will have two turtles, pointing at right angles to each other. To see that, try this:

(draw 100)

You will see two lines. Now, if you evaluate those two expression again, you will have four turtles, etc

• (split* E ...) is similar to (split E ...), except it creates as many turtles as there are expressions and each turtles does one of the expression. For example, to create two turtles, one pointing at $\pi/2$ and one at $\pi/3$, evaluate this:

3.1. Turtles 3. Turtles

• (tprompt E...) provides a way to limit the splitting of the turtles. Before the expression E is run, the state of the turtles (how many, their positions and headings) is "checkpointed," then E is evaluated and the state of the turtles is restored, but all drawing that may have occurred during execution of E remains.

For example, if you do this:

```
(tprompt (draw 100))
```

the turtle will move forward 100 pixels, draw a line there and then be immediately put back in it's original position. Also, if you do this:

the turtle will split into two turtles, one will turn 90 degrees and then the turtles will be put back into their original state – as if the split never took place.

The fern functions below demonstrate more advanced use of **tprompt**.

In the file **turtles-examples.ss** in the **graphics** library of your PLT distribution, you will find these functions and values defined, as example turtle programs. (The file is located in the **graphics** subdirectory of the **collects** subdirectory of the PLT distribution).

- (regular-poly sides radius) draws a regular poly centered at the turtle with sides sides and with radius radius.
- (regular-polys sides s) draws s regular polys spaced evenly outwards with sides sides.
- (radial-turtles n) places 2ⁿ turtles spaced evenly pointing radially outward
- (spaced-turtles n) places 2^n turtles pointing in the same direction as the original turtle evenly spaced in a line.
- (spokes) draws some spokes, using radial-turtles and spaced-turtles
- (spyro-gyra) draws a spyro-grya reminiscent shape
- (neato) as the name says...
- (graphics-bexam) draws a fractal that came up on an exam I took.
- serp-size a constant which is a good size for the serp procedures
- (serp serp-size), (serp-nosplit serp-size) draws the Serpinski triangle in two different ways, the first using split heavily. After running the first one, try executing (draw 10).
- koch-size a constant which is a good size for the koch procedures
- (koch-split koch-size),(koch-draw koch-size) draws the same koch snowflake in two different ways.
- (lorenz a b c) watch the lorenz "butterfly" attractor with initial values a b and c.
- (lorenz1) a good setting for the lorenz attractor
- (peano1 peano-size)

This will draw the Peano space-filling curve, using split.

3. Turtles 3.2. Value Turtles

• (peano2 peano-size)

This will draw the Peano space-filling curve, without using split.

- fern-size a good size for the fern functions
- (fern1 fern-size) You will probably want to point the turtle up before running this one, with something like:

• (fern2 fern-size) a fern - you may need to backup a little for this one.

3.2 Value Turtles

There are two ways to use the turtles in DrScheme. You can use it as a TeachPack (see the DrScheme manual for details of TeachPacks) or as a library. Use the **value-turtles.ss** TeachPack.

In the MrEd language or in a module, load turtles with

```
(require (lib "value-turtles.ss" "graphics"))
```

The value turtles are a variation on the turtles library. Rather than having just a single window where each operation changes the state of that window, in this library, the entire turtles window is treated as a value. This means that each of the primitive operations accepts, in addition to the usual arguments, a turtles window value and instead of returning nothing, returns a turtles window value.

The following are the value turtle functions:

- (turtles number number [number number]) creates a new turtles window. The first two arguments are the width and height of the turtles window. The remaining arguments specify the x,y position of the initial turtle and the angle. The default to a turtle in the middle of the window, pointing to the right.
- (move n turtles) moves the turtle n pixels, returning a new turtles window.
- (draw n turtles) moves the turtle n pixels and draws a line on that path, returning a new turtles window.
- (erase n turtles) moves the turtle n pixels and erases along that path, returning a new turtles window.
- (move-offset h v turtles), (draw-offset h v turtles), (erase-offset h v turtles) are just like move, draw and erase, except they take a horizontal and vertical offset from the turtle's current position.
- (turn theta turtles) turns the turtle theta degrees counter-clockwise, returning a new turtles window.
- (turn/radians theta) turns the turtle theta radians counter-clockwise, returning a new turtles window.
- (merge turtles turtles)

The split and tprompt functionality provided by the imperative turtles implementation aren't needed for this, since the turtles window is itself a value.

Instead, the merge accepts two turtles windows and combines the state of the two turtles windows into a single window. The new window contains all of the turtles of the previous two windows, but only the line drawings of the first turtles argument.

3.2. Value Turtles 3. Turtles

In the file **value-turtles-examples.ss** in the **graphics** library of your PLT distribution, you will find these functions and values defined, as example turtle programs. (The file is located in the **graphics** subdirectory of the **collects** subdirectory of the PLT distribution).

It contains a sampling of the examples from the normal turtles implementation, but translated to use merge and the values turtles.

\mathbf{Index}

#/return, 9	$\mathtt{get} ext{-}\mathtt{color} ext{-}\mathtt{pixel}, rac{3}{}$
D 11 (1 A) 1 10	$\mathtt{get} ext{-}\mathtt{key} ext{-}\mathtt{press}, extstyle{9}$
Butterfly Attractor, 12	get-mouse-click, 8
change-color 3	get-pixel, 3
change-color, 3	get-string-size, 7
clear, 11	graphics
clear-ellipse, 5	simple, $\frac{2}{}$
clear-line, 4	graphics.ss, 2
clear-pixel, 4	$graphics:posn-less^2$, 9
clear-polygon, 6	$graphics:posn^{*}, 9$
clear-rectangle, 4	graphics@, 10
clear-solid-ellipse, 5	${ t graphics \^{ t , 9}}$
clear-solid-polygon, 6	
clear-solid-rectangle, 5	key-value, 9
clear-string, 6	Koch Snowflake, 12
clear-viewport, 4	
close-graphics, 2	left-mouse-click?, 9
close-viewport, 2	Lorenz Attractor, 12
copy-viewport, 4	
default-display-is-color?, 3	make-posn, 3
draw, 11, 13	make-rgb, 3
draw-ellipse, 5	merge, 13
draw-line, 4	middle-mouse-click?, 9
draw-offset, 11, 13	mouse-click-posn, 9
draw-pixel, 4	move, 11, 13
draw-pixmap, 7	$\mathtt{move-offset}, \textcolor{red}{11}, \textcolor{red}{13}$
draw-pixmap-posn, 6	anan maankiaa O
draw-polygon, 6	open-graphics, 2
draw-rectangle, 4	open-pixmap, 2
draw-solid-ellipse, 5	open-viewport, $\frac{2}{2}$
draw-solid-polygon, 6	Peano space-filling curve, 12
draw-solid-rectangle, 5	posn-x, 3
draw-string, 6	-
draw-viewport, 4	$posn?, \frac{3}{3}$
diaw viewpoit, 4	query-mouse-posn, 9
erase, 11, 13	query mouse posis, o
erase-offset, 11, 13	ready-key-press, 9
, ,	ready-mouse-click, 8
Fern Fractal, 13	ready-mouse-release, 8
flip-ellipse, 5	rgb-red, 3
flip-line, 4	rgb?, 3
flip-pixel, 4	right-mouse-click?, 9
flip-polygon, 6	3
flip-rectangle, 5	save-pixmap, 7
flip-solid-ellipse, 5	Serpinski Triangle, 12
flip-solid-polygon, 6	split, 11
flip-solid-rectangle, 5	split*, 11
flip-string, 6	•
flip-viewport, 4	test-pixel, $\frac{3}{}$

```
tprompt, 12
turn, 11, 13
Turtles
    Value, 13
turtles, 11, 13
turtles.ss, 11

Value Turtles, 13
value-turtles.ss, 13
viewport, 2
viewport->snip, 7
viewport-dc, 7
viewport-flush-input, 9
viewport-offscreen-dc, 7
```