Chapter 2. Scheme Extensions Ga thru Uz

Topic:	Ignore

grid:display–color

Scheme Extension:	Grids, Colors
Action:	Gets the grid display color.
Filename:	gi/gi_scm/grid_scm.cxx
APIs:	None
Syntax:	(**grid:display–color**)
Arg Types:	None
Returns:	color
Errors:	None
Description:	Returns the grid color value.
Limitations:	None

Example:

```
; grid:display–color
; Set the grid display color.
(grid:set–display–color 5)
;; ()
; Get the grid display color.
(grid:display–color)
;; #[color 1 1 0]
; Turn the grid on.
(grid:set–display #t)
;; ()
```

grid:display–interval

Scheme Extension:	Grids
Action:	Gets the display interval for the grid.

Filename: gi/gi_scm/grid_scm.cxx

APIs: api_gi_get_grid_display_intervals

Syntax: (**grid:display-interval**)

Arg Types: None

Returns: integer ...

Errors: None

Description: This extension returns the *x* and *y* grid display intervals as a list of two integers. The default interval is 10 in each direction.

Limitations: None

Example:

```
; grid:display-interval
; Turn the grid on.
(grid:set-display #t)
;; ()
; Set the grid display interval.
(grid:set-display-interval 8 8)
;; ()
; Get the grid display interval.
(grid:display-interval)
;; (8 8)
```

grid:display–method

Scheme Extension: Grids

Action: Gets the grid display method.

Filename: gi/gi_scm/grid_scm.cxx

APIs: api_gi_get_grid_display_method

Syntax: (**grid:display-method**)

Arg Types: None

Returns: string

Errors: None

Description: This extension returns a string indicating the current display method of the grid (“.” or “–”).

Limitations: None

Example:
```
; grid:display-method
; Turn the grid on.
(grid:set-display #t)
;; ()
; Set the grid display method.
(grid:set-display-method ”-”)
;; ()
; Get the grid display method.
(grid:display-method)
;; ”-”
; Set the grid display method.
(grid:set-display-method ”.”)
;; ()
```

grid:displayed?

Scheme Extension: Grids

Action: Determines if the grid is displayed.

Filename: gi/gi_scm/grid_scm.cxx

APIs: None

Syntax: (**grid:displayed?**)

Arg Types: None

Returns: boolean

Errors: None

Description: This extension returns #t if the grid is displayed; otherwise, it returns #f. By default, the grid is not displayed.

Limitations: None

Example:
```
; grid:displayed?
; Determine if the grid is displayed.
(grid:displayed?)
;; #f
; Turn the grid on.
(grid:set-display #t)
;; ()
; Determine if the grid is displayed.
(grid:displayed?)
;; #t
```

grid:interval

Scheme Extension: Grids

Action: Gets the snap interval between displayed grid points.

Filename: gi/gi_scm/grid_scm.cxx

APIs: api_gi_get_grid_snap_intervals

Syntax: (**grid:interval**)

Arg Types: None

Returns: real ...

Errors: None

Description: This extension returns the grid's dx, dy, and dz as a list of three reals. The reals indicates the distance between neighboring grid lines in the *x*–, *y*–, and *z*–directions. The default is 1.0 for *x* and *y*, and 0.0 for *z*.

Limitations: None

Example:

```
; grid:interval
; Turn the grid on.
(grid:set–display #t)
;; ()
; Set the grid display interval.
(grid:set–interval 2 2)
;; ()
; Get the grid display interval.
(grid:interval)
;; (2 2 0)
```

grid:set–display

Scheme Extension: Grids

Action: Sets the grid display on or off.

Filename: gi/gi_scm/grid_scm.cxx

APIs: api_gi_display_grid

Syntax: (**grid:set–display** on–off)

Arg Types:

on–off	boolean

Returns:	unspecified
Errors:	None
Description:	A setting of #t activates the grid display, which is a grid of dots or dashes (depending on grid:set–display–method) spaced at intervals established by grid:set–interval. The grid is initialized off; #f deactivates the grid display.
Limitations:	None

Example:

```
; grid:set-display
; Turn on display of the grid.
(grid:set-display #t)
;; ()
; Turn the grid off.
(grid:set-display #f)
;; ()
```

grid:set–display–color

Scheme Extension: Grids, Colors

Action:	Sets the grid display color.	
Filename:	gi/gi_scm/grid_scm.cxx	
APIs:	None	
Syntax:	(**grid:set–display–color** color)	
Arg Types:	color	color
Returns:	unspecified	
Errors:	None	

Description: color specifies the desired color for the grid display, and the grid immediately changes to the selected color. The initial setting is 7. color values include:

0	= Black
1	= Red
2	= Green
3	= Blue
4	= Cyan
5	= Yellow
6	= Magenta
7	= White

Limitations: None

Example:
```
; grid:set-display-color
; Turn the grid on.
(grid:set-display #t)
;; ()
; Set the display color of the grid as an integer.
(grid:set-display-color 5)
;; ()
; Set the display color of the grid as an rgb value.
(grid:set-display-color (color:rgb 0.66 0.54 0.9))
;; ()
```

grid:set–display–interval

Scheme Extension: Grids

Action: Sets the grid display interval.

Filename: gi/gi_scm/grid_scm.cxx

APIs: api_gi_set_grid_display_intervals

Syntax: (**grid:set-display-interval** nx=10 ny=10)

Arg Types:

nx	integer
ny	integer

Returns: unspecified

Errors: None

Description: nx specifies the *x*–direction factor grid spacing and its default value is 10. ny specifies the *y*–direction factor grid spacing and is initialized at 10. If nx or ny is 1, every grid point displays. If nx or ny is 2, every other or every second grid point displays. The system automatically blanks the grid if the user tries to display an excessive amount of grid intervals.

Limitations: None

Example:
```
; grid:set-display-interval
; Turn the grid on.
(grid:set-display #t)
;; ()
; Set the grid display interval.
(grid:set-display-interval 1 1)
;; ()
```

grid:set–display–method

Scheme Extension: Grids

Action: Sets the grid display method.

Filename: gi/gi_scm/grid_scm.cxx

APIs: api_gi_set_grid_display_method

Syntax: (**grid:set–display–method** method=dots)

Arg Types:

method	string

Returns: unspecified

Errors: None

Description: method specifies the type of grid display. The grid immediately changes to the selected method. method values are "." for dots (the default) and "–" for dashed lines.

Limitations: None

Example:

```
; grid:set–display–method
; Set the grid display method to dots.
(grid:set–display–method ".")
;; ()
; Set the grid display method to dashes.
(grid:set–display–method "–")
;; ()
```

grid:set–interval

Scheme Extension: Grids

Action: Sets the interval between grid points for display and snapping.

Filename: gi/gi_scm/grid_scm.cxx

APIs: api_gi_set_grid_snap_intervals

Syntax: (**grid:set–interval** dx dy [dz])

Arg Types:

dx	real
dy	real
dz	real

Returns: unspecified

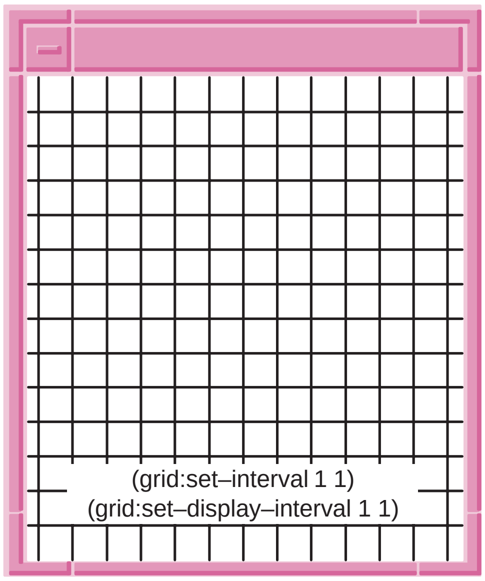
Errors: None

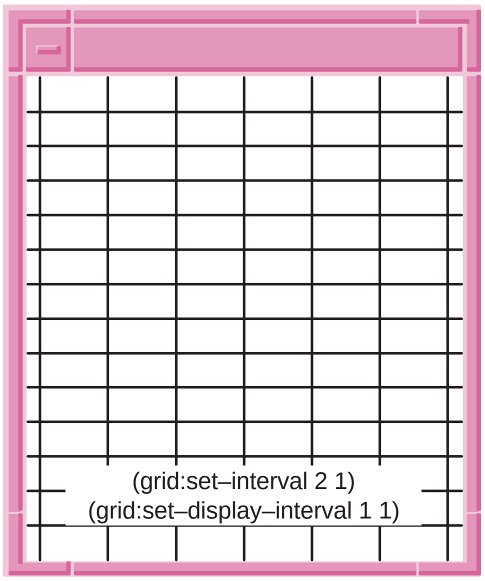
Description: dx specifies the *x*–direction distance interval and is initialized to 1. dy specifies the *y*–direction distance interval and is initialized to 1. The optional dz specifies the *z*–direction distance interval and is initialized to 0.

Limitations: None

Example:

```
; grid:set-interval
; Turn the grid on.
(grid:set-display #t)
;; ()
; Set the grid display interval to 3 x 3.
(grid:set-interval 3 3)
;; ()
; Set the grid display interval to 2 x 2.
(grid:set-interval 5 5)
;; ()
; In the following figures:
; dashed lines indicate that they are
; not displayed but are still available
; as snap points.
```





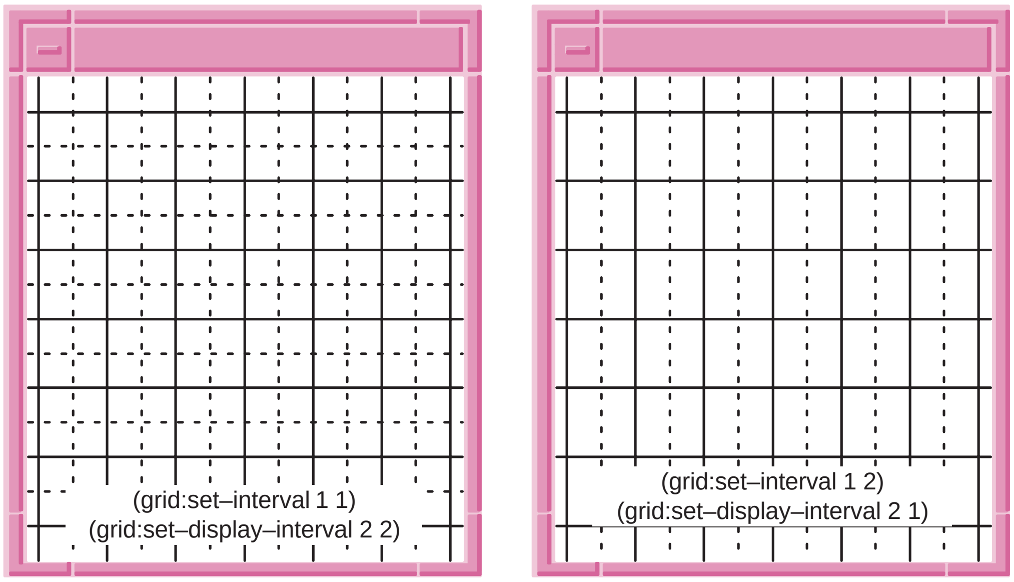

(grid:set–interval 1 1)
(grid:set–display–interval 2 2)
(grid:set–interval 1 2)
(grid:set–display–interval 2 1)

Figure 2-1. grid:set–Interval

grid:set–snapping

Scheme Extension:	Grids, Picking
Action:	Sets grid snapping on or off.
Filename:	gi/gi_scm/grid_scm.cxx
APIs:	api_gi_activate_grid_snap
Syntax:	`(`**`grid:set–snapping`**` on–off)`
Arg Types:	on–off boolean
Returns:	unspecified
Errors:	None
Description:	When grid snapping is on, positions computed by a pick position snap to the nearest grid point. off deactivates grid snapping; the default is off.
Limitations:	None

Example:

```
; grid:set-snapping
; Turn the grid on.
(grid:set-display #t)
;; ()
; Enable snapping of pick positions to the
; nearest grid point.
(grid:set-snapping #t)
;; ()
; Define the first corner of a rectangle.
(define start (pick:position (read-event)))
;; start
; Create a rectangle rubberband driver.
(rbd:rectangle #t start)
;; #[rbd-driver 401b1df0]
; Define the second corner of a rectangle.
(define end (pick:position (read-event)))
;; end
; Define the rectangle.
(define rectangle
    (lambda (start end)
        (let ((x1 (position:x start))
            (x2 (position:x end))
            (y1 (position:y start))
            (y2 (position:y end))
            (z1 (position:z start))
            (z2 (position:z end)))
    (list (edge:linear start end))
    (let ((corner1 (position x2 y1 z2))
        (corner2 (position x1 y2 z1)))
    (list (edge:linear start corner1)
        (edge:linear corner1 end)
        (edge:linear end corner2)
        (edge:linear corner2 start))))))
;; rectangle
; Draw the rectangle.
(rectangle start end)
; Draw the diagonal.
(define linear (edge:linear start end))
;; linear
; Turn off the rubberband driver.
(rbd:remove-type rbd:rectangle?)
;; ()
```

grid:snapping–on?

Scheme Extension: Grids, Picking

Action:	Determines the current status of grid snapping.
Filename:	gi/gi_scm/grid_scm.cxx
APIs:	None
Syntax:	(**grid:snapping–on?**)
Arg Types:	None
Returns:	boolean
Errors:	None
Description:	Refer to Action.
Limitations:	None

Example:

```
; grid:snapping–on?
; Determine if snapping of pick positions
; to the nearest grid point is enabled.
(grid:snapping–on?)
;; #f
; Enable snapping of pick positions
; to the nearest grid point.
(grid:set–snapping #t)
;; ()
(grid:snapping–on?)
;; #t
```

rbd:add

Scheme Extension: Rubberbanding

Action:	Adds a rubberband driver to the list of active drivers.	
Filename:	gi/gi_scm/rb_scm.cxx	
APIs:	None	
Syntax:	(**rbd:add** driver)	
Arg Types:	driver	rbd–driver | (rbd–driver ...)
Returns:	(rbd–driver ...)	

Errors: driver is the wrong type.

Description: This extension adds the given rubberband driver(s) to the list of active drivers. The active list is the rubberband driver list at the top of the stack. Adding a rubberbander to the list activates it.

Limitations: None

Example:

```
; rbd:add
; Create and activate a line rubberband anchored at
; position 0 0 0
(define rubber1 (rbd:line #t (position 0 0 0)))
;; rubber1
; Create a line rubberband anchored at
; position 20 10 0, but do not activate it.
(define rubber2 (rbd:line #f (position 20 10 0)))
;; rubber2
; Add second rubberband driver to the active list.
(rbd:add rubber2)
;; (#[rbd–driver 401b1018] #[rbd–driver 401b1026])
; Get the list of active rubberband drivers.
(rbd:drivers)
;; (#[rbd–driver 401b1018] #[rbd–driver 401b1026])
; Clear the active driver list.
(rbd:clear #f)
;; ()
(rbd:clear #f)
;; ()
```

rbd:clear

Scheme Extension: Rubberbanding

Action: Removes last rubberband driver from active driver list, or removes all drivers from stack of driver lists.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: `(`**`rbd:clear`**` [flag–all=#t])`

Arg Types: flag–all boolean

Returns: unspecified

Errors: None

Description: This clears the entire rubberband driver stack or only the last driver added to the active driver list. The active list is the rubberband driver list at the top of the stack. If flag–all is #t, which is the default, this extension clears all drivers from all levels of the stack of lists of rubberband drivers. If flag–all is #f, only the top of the stack is affected; the last driver added to the list is removed.

Limitations: None

Example:

```
; rbd:clear
; Add a rubberband driver to the active list.
(rbd:add (rbd:rectangle #f (position 0 0 0)))
;; (#[rbd-driver 4022a2f8])
; Add a rubberband driver to the active list.
(rbd:add (rbd:line #f (position 10 20 30)))
;; (#[rbd-driver 4022a3c0] #[rbd-driver 4022a2f8])
; Clear the last defined rubberbanding procedure.
(rbd:clear #f)
;; ()
; Clear the first defined rubberbanding procedure.
(rbd:clear #f)
;; ()
```

rbd:drag

Scheme Extension: Rubberbanding

Action: Creates a drag rubberband driver and, optionally, makes it active.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: `(`**`rbd:drag`**` activate entity-list [base=position 0 0 0])`

Arg Types:

activate	boolean
entity–list	entity | (entity ...)
base	position

Returns: rbd–driver

Errors: None

Description: If activate is #t, this extension automatically adds the new driver to the list of active drivers; otherwise, the extension creates and returns it. The argument entity–list specifies a list of entities to be dragged. The argument base specifies a position from which cursor movements are measured; the default is (position 0 0 0).

The drag rubberband driver displays a white wireframe view of the entities that move with the cursor.

Dragging only occurs parallel to the rubberbanding projection plane. When rb–position–hook is set, this is the plane onto which mouse positions are projected. The default is the *xy*–plane of the active coordinate system.

Limitations: None

Example:

```
; rbd:drag
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 20 15)))
;; block1
; Enable rubberband dragging.
(rbd:drag #t block1)
;; #[rbd–driver 401ca0a8]
; Clear the first defined rubberbanding procedure.
(rbd:clear #f)
;; ()
; Double–click entity dragging example follows.
; Create a user interface error dialog box.
(ui:error–dialog
        ”Click on block1 entity.”)
; Click on block entity.
(define evt (read–event))
;; evt
; Get entity portion of event.
(define ent (pick:entity evt))
;; ent
; Test to see if object is an entity.
; Get original location of entity/event.
(define pos1 (pick:position evt))
;; pos1
; Test to see if object is entity.
; Move mouse to new position. Image should follow.
; Drag entity to new location.
; Click on where entity is to go.
(if (entity? ent)
    (begin
        (rbd:drag #t ent pos1)
        (ui:error–dialog
            ”Click on Destination for block1.”)
        (define pos2 (pick:position (read-event)))
        (rbd:remove-type rbd:drag?)
        (entity:transform ent
            (transform:translation
            (gvector:from-to pos1 pos2))))
        (ui:error-dialog
            ”Object selected is not entity.”))
;; #[entity 2 1]
; Move entity back to where it originally was.
(if (entity? ent)
    (begin
        (entity:transform ent
            (transform:translation
            (gvector:from-to pos2 pos1))))
    (ui:error-dialog
        ”Object selected is not entity.”))
;; #[entity 2 1]
```

rbd:drag?

Scheme Extension:	Rubberbanding
Action:	Determines if a drag rubberband driver is in the active driver list.
Filename:	gi/gi_scm/rb_scm.cxx
APIs:	None
Syntax:	(**rbd:drag?** object)
Arg Types:	object scheme–object
Returns:	boolean
Errors:	None
Description:	This extension returns #t if a drag rubberband driver in in the active driver list (e.g., list at top of the driver stack); otherwise, it returns #f. Used primarily with the rbd:remove–type extensions.
Limitations:	None

Example:

```
; rbd:drag?
; Determine if the Boolean is a rbd–drag driver.
(rbd:drag? #t)
;; #f
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Determine if the rbd–drag driver is actually
; a rbd–drag driver.
(rbd:drag? (rbd:drag #t block1 (position 1 2 3)))
;; #t
; Determine if the rbd–line driver is a
; rbd–drag driver.
(rbd:drag? (rbd:line #t (position 1 2 3)))
;; #f
; Stop the dragging process.
(rbd:clear #f)
;; ()
(rbd:clear #f)
;; ()
```

rbd:drivers

Scheme Extension: Rubberbanding

Action: Gets a list of active rubberband drivers at the top of the stack or at a specific stack level.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: (**rbd:drivers** [depth])

Arg Types: depth integer

Returns: (rbd–driver ...)

Errors: None

Description: This extension returns a list of rubberband drivers from a stack of lists of rbd–drivers. The argument depth specifies the level in the stack for the driver. A depth of 0 is the top of the stack. If it is specified, it must be positive.

Limitations: None

Example:

```
; rbd:drivers
; Create a line rubberband driver.
(define rubber1 (rbd:line #t (position 0 0 0)))
;; rubber1
(define rubber2 (rbd:line #f (position 0 0 0)))
;; rubber2
; Add second rbd to active list.
(rbd:add rubber2)
;; (#[rbd-driver 402274e8] #[rbd-driver 402273f0])
; Push the combined list on the stack and make
; first rbd active at top of stack.
(rbd:push rubber1)
;; (#[rbd-driver 402273f0])
; Push the first on the stack and make
; second rbd active at top of stack.
(rbd:push rubber2)
;; (#[rbd-driver 402274e8])
; Get a list of rubberband-drivers from the stack
; of drivers.
(rbd:drivers 1)
;; (#[rbd-driver 402273f0])
(rbd:drivers 2)
;; (#[rbd-driver 402274e8] #[rbd-driver 402273f0])
(rbd:drivers)
;; (#[rbd-driver 402274e8])
; Pop the top of a stack of rubberband-driver lists.
(rbd:pop)
;; (#[rbd-driver 402274e8])
(rbd:pop)
;; (#[rbd-driver 402273f0])
(rbd:drivers)
;; (#[rbd-driver 402274e8] #[rbd-driver 402273f0])
(rbd:clear)
;; ()
```

rbd:generic

Scheme Extension: Rubberbanding

Action: Creates a generic rubberband driver, makes it active, and gives status report in command window.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: `(`**`rbd:generic`**` activate base=position 0 0 0)`

Arg Types:	activate	boolean
	base	position

Returns: rbd–driver

Errors: None

Description: The argument base specifies the start position of the line. If base is not specified, the default start position is (0, 0, 0). If activate is #t, this extension automatically adds the new driver to the list of active drivers; otherwise, this extension creates and returns it.

The generic rubberband driver displays a report in the command window of the calls made to the driver's hook procedures. This extension writes the event type and the coordinates in pixels to the text output window.

Limitations: None

Example:

```
; rbd:generic
; Create a generic rubberband driver.
(rbd:generic #t (position 3 –5 20))
;; #[rbd–driver 401b1018]
; The system activates the rubberband driver.
; The report of mouse information is displayed
; in the command window.
; Clear this rubberbander from the active list
(rbd:clear #t)
;; ()
```

rbd:generic?

Scheme Extension: Rubberbanding

Action: Determines if a generic rubberband driver is in the active driver list.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: `(`**`rbd:generic?`**` object)`

Arg Types:	object	scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if a generic rubberband driver is in the list of active rubberband drivers (e.g., the list at the top of the stack); otherwise, it returns #f.

Limitations: None

Example:

```
; rbd:generic?
; Determine if the Boolean is a rbd–generic driver.
(rbd:generic? #t)
;; #f
; Determine if the rbd–generic driver is actually
; an rbd–generic driver.
(rbd:generic? (rbd:generic #t (position 1 2 3)))
;; #t
; Determine if the rbd–line driver is an
; rbd–generic driver.
(rbd:generic? (rbd:line #t (position 1 2 3)))
;; #f
; Clear this rubberbander from the active list
(rbd:clear #t)
;; ()
```

rbd:line

Scheme Extension: Rubberbanding

Action: Creates a line rubberband driver and, optionally, makes it active.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: `(`**`rbd:line`**` activate [base=position 0 0 0])`

Arg Types:

activate	boolean
base	position

Returns: rbd–driver

Errors: None

Description: The argument base specifies the start position of the line. If base is not specified, the default start position is (0, 0, 0). If activate is #t, this extension automatically adds the new driver to the list of active drivers; otherwise, this extension creates and returns the new driver.

The line rubberband driver displays a white line with one end anchored at the base position and the other end following the position of the cursor. The moving end of the rubberband line is located on the rubberbanding projection plane. Mouse positions are projected onto this plane during rubberbanding as a result of setting the hook procedure,rb–position–hook. If this hook procedure is set to the empty list (default), then the projection plane is the *xy* plane of the active coordinate system.

Limitations: None

Example:
```
; rbd:line
; Define the position of the start of line.
(define start (pick:position (read–event)))
;; start
; Create a line rubberband driver.
(rbd:line #t start)
;; #[rbd–driver 401b1df0]
; Define the position of the end of line.
(define end (pick:position (read–event)))
;; end
; Define the line.
(define line (edge:linear start end))
;; line
; Turn off rbd.
(rbd:remove–type rbd:line?)
;; ()
```

rbd:line?

Scheme Extension: Rubberbanding

Action: Determines if a line rubberband driver is in the active driver list.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: (**rbd:line?** object)

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if a line rubberband driver is in the list of active rubberband drivers (e.g., the list at the top of the stack); otherwise, it returns #f.

Limitations: None

Example:
```
; rbd:line?
; Determine if the Boolean is an rbd–line driver.
(rbd:line? #t)
;; #f
; Define the position of the start of line.
(define start (pick:position (read–event)))
;; start
; Create a line rubberband driver.
(rbd:line #t start)
;; #[rbd–driver 401b1df0]
; Define the position of the end of line.
(define end (pick:position (read–event)))
;; end
; Define the line.
(define line (edge:linear start end))
;; line
; Turn off rbd.
(rbd:remove–type rbd:drag?)
;; (#[rbd–driver 401e7530])
(rbd:remove–type rbd:line?)
;; ()
```

rbd:pop

Scheme Extension: Rubberbanding

Action: Pops the top of a stack of lists of rubberband drivers.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: (**rbd:pop**)

Arg Types: None

Returns: (rbd–driver ...) **|** boolean

Errors: None

Description: This extension deactivates the current rubberband driver list by removing it from the top of the stack. If there was a prior list that had been pushed onto the stack, it becomes the active rubberband driver list. If the stack was empty, this returns #f. Returning #f is not the same as returning an empty list. #f indicates the stack is empty. An empty list indicates that the top list on the stack had no active drivers; there could be other drivers still in the rubberband stack.

Limitations: None

Example:
```
; rbd:pop
; Create a line rubberband driver.
(define rubber1 (rbd:line #t (position 0 0 0)))
;; rubber1
(define rubber2 (rbd:line #f (position 10 20 0)))
;; rubber2
; Add second rbd to active list.
(rbd:add rubber2)
;; (#[rbd–driver 40227478] #[rbd–driver 40227380])
; Push the combined list on the stack and make
; the first rbd active at top of stack.
(rbd:push rubber1)
;; (#[rbd–driver 40227380])
; Push the first on the stack and make
; the second rbd active at top of stack.
(rbd:push rubber2)
;; (#[rbd–driver 40227478])
; Get a list of rubberband–drivers from the stack of
; drivers.
; Lowest in the stack
(rbd:drivers 2)
;; (#[rbd–driver 40227478] #[rbd–driver 40227380])
; Middle of the stack.
(rbd:drivers 1)
;; (#[rbd–driver 40227380])
; Top of the stack.
(rbd:drivers)
;; (#[rbd–driver 40227478])


; Pop the top of a stack of rubberband–driver lists.
(rbd:pop)
;; (#[rbd–driver 40227478])
(rbd:pop)
;; (#[rbd–driver 40227380])
(rbd:drivers)
;; (#[rbd–driver 40227478] #[rbd–driver 40227380])
(rbd:clear)
;; ()
```

rbd:push

Scheme Extension: Rubberbanding

Action:	Pushes a new list of drivers onto the stack of driver lists.
Filename:	gi/gi_scm/rb_scm.cxx
APIs:	None
Syntax:	(**rbd:push** [driver])
Arg Types:	driver — rbd–driver | (rbd–driver ...)
Returns:	(rbd–driver ...)
Errors:	driver is the wrong type.

Description: This extension deactivates the current rubberband driver list by pushing it down into the stack. If an optional rubberband driver list was specified, it becomes the active driver list. If no optional rubberband driver list is specified, then the active rubberband driver is an empty list.

The list of drivers pushed onto the stack can be retrieved and reactivated by using rbd:pop to pop the current driver list off of the stack.

Limitations: None

Example:

```
; rbd:push
; Create a line rubberband driver.
(define rubber1 (rbd:line #t (position 0 0 0)))
;; rubber1
(define rubber2 (rbd:line #f (position 10 20 0)))
;; rubber2
; Add second rbd to active list.
(rbd:add rubber2)
;; (#[rbd-driver 402274e8] #[rbd-driver 402273f0])
; Push the combined list on the stack and make
; the first rbd active at top of stack.
(rbd:push rubber1)
;; (#[rbd-driver 402273f0])
; Push the first on the stack and make the
; second rbd active at top of stack.
(rbd:push rubber2)
;; (#[rbd-driver 402274e8])
; Get a list of rubberband-drivers from the stack
; of drivers.
; Lowest in the stack.
(rbd:drivers 2)
;; (#[rbd-driver 402274e8] #[rbd-driver 402273f0])
; Middle of the stack.
(rbd:drivers 1)
;; (#[rbd-driver 402273f0])
; Top of the stack.
(rbd:drivers)
;; (#[rbd-driver 402274e8])
; Pop the top of a stack of rubberband-driver lists.
(rbd:pop)
;; (#[rbd-driver 402274e8])
(rbd:pop)
;; (#[rbd-driver 402273f0])
(rbd:drivers)
;; (#[rbd-driver 402274e8] #[rbd-driver 402273f0])
(rbd:clear)
;; ()
```

rbd:rectangle

Scheme Extension: Rubberbanding

Action: Creates a rectangle rubberband driver and, optionally, makes it active.

Filename: gi/gi_scm/rb_scm.cxx

APIs:	None	
Syntax:	(**rbd:rectangle** activate [base=position 0 0 0])	
Arg Types:	activate	boolean
	base	position
Returns:	rbd–driver	
Errors:	None	

Description: The argument base specifies a starting corner position of the rectangle. If base is not specified, the default start position is (0, 0, 0). If activate is #t, this extension automatically adds the new driver to the list of active drivers; otherwise, this extension creates the driver list and returns it.

The rectangle rubberband driver displays a white rectangle with one corner anchored at the base position and the other corner following the position of the cursor. The rectangle is in a plane parallel to the *xy* plane of the active WCS.

The moving end of the rubberband line is located on the rubberbanding projection plane. Mouse positions are projected onto this plane during rubberbanding as a result of setting the hook procedure,rb–position–hook. If this hook procedure is set to the empty list (default), then the projection plane is the *xy* plane of the active coordinate system.

Limitations: None

Example:

```
; rbd:rectangle
; Define the first corner of the rectangle.
(define start (pick:position (read–event)))
; Using the left mouse button, select a position
; in the active view on the screen.
;; start
; Create a rectangle rubberband driver.
(rbd:rectangle #t start)
;; #[rbd–driver 401b1df0]
; Define the second corner of the rectangle.
(define end (pick:position (read–event)))
; Using the left mouse button, select a
; diagonal position in the active view.
;; end
```

```
; Define the rectangle.
(define rectangle
    (lambda (start end)
        (let ((x1 (position:x start))
            (x2 (position:x end))
            (y1 (position:y start))
            (y2 (position:y end))
            (z1 (position:z start))
            (z2 (position:z end)))
    (list (edge:linear start end))
    (let ((corner1 (position x2 y1 z2))
        (corner2 (position x1 y2 z1)))
    (list (edge:linear start corner1)
        (edge:linear corner1 end)
        (edge:linear end corner2)
        (edge:linear corner2 start))))))
;; rectangle
; Draw the rectangle.
(rectangle start end)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;; #[entity 6 1])
; Turn off rbd.
(rbd:remove–type rbd:rectangle?)
;; ()
```

rbd:rectangle?

Scheme Extension: Rubberbanding

Action:	Determines if a rectangle rubberband driver is in the active driver list.
Filename:	gi/gi_scm/rb_scm.cxx
APIs:	None
Syntax:	(**rbd:rectangle?** object)
Arg Types:	object scheme–object
Returns:	boolean
Errors:	None
Description:	This extension returns #t if there is a rectangle rubberband driver in the active driver list (e.g., list at the top of the rubberband stack). Otherwise, this returns #f.

Limitations: None

Example:

```
; rbd:rectangle
; Determine if the Boolean is a rbd–rectangle driver.
(rbd:rectangle? #t)
;; #f
; Determine if the rbd–rectangle driver is actually
; a rbd–rectangle driver.
(rbd:rectangle? (rbd:rectangle #t (position 0 30 0)))
;; #t
; Determine if the rbd–line driver is a
; rbd–rectangle driver.
(rbd:rectangle? (rbd:line #t (position 1 2 3)))
;; #f
; Clear the drivers.
(rbd:clear #f)
;; ()
(rbd:clear #f)
;; ()
```

rbd:remove

Scheme Extension: Rubberbanding

Action: Removes most recently added driver(s) from the list of active drivers.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: (**rbd:remove** driver)

Arg Types: driver rbd–driver | (rbd–driver ...)

Returns: (rbd–driver ...)

Errors: driver is the wrong type.

Description: This extension removes the most recently added driver from the list of active rubberband drivers (e.g., the list at the top of a stack). This extension returns the list of active drivers.

Limitations: None

Example:

```
; rbd:remove
; Remove a specific driver from the list of
; active drivers.
; Define a rbd–line driver.
(define rb–driver (rbd:line #t (position 10 5 0)))
;; rb–driver
; Define a rectangle.
(define rectangle (rbd:rectangle #t))
;; rectangle
(rbd:drivers)
;; (#[rbd–driver 402276a0] #[rbd–driver 40227608])
; Remove the rbd–line driver from the active list.
(rbd:remove rb–driver)
;; (#[rbd–driver 402276a0])
(rbd:drivers)
;; (#[rbd–driver 402276a0])
; Clear the last driver.
(rbd:clear #f)
;; ()
```

rbd:remove–type

Scheme Extension: Rubberbanding

Action: Removes the most recently added rubberband driver(s) matching the given type predicate from the list of active drivers.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: (**rbd:remove–type** type–predicate)

Arg Types: type–predicate scheme–procedure

Returns: (rbd–driver ...)

Errors: type–predicate is the wrong argument type.

Description: This extension removes the most recently added rubberband driver(s) matching the given type–predicate from the list of active rubberband drivers. type–predicate is a Scheme procedure that takes one argument and returns a boolean. If more than one driver matches the type–predicate, this extension removes the most recently added match. If no drivers match, this extension returns a list of the currently active drivers and the list is active drivers is unchanged.

Limitations: None

Example:

```
; rbd:remove–type
; Create a line rubberband driver.
(rbd:line #t (position 10 20 0))
;; #[rbd–driver 40228d88]
; Create a rectangular rubberband driver.
(rbd:rectangle #t (position 0 30 0))
;; #[rbd–driver 40228eb8]
; Create another line rubberband driver.
(rbd:line #t (position 20 5 0))
;; #[rbd–driver 40228fe0]
; List the rubberband drivers.
(rbd:drivers)
;; (#[rbd–driver 40228fe0] #[rbd–driver 40228eb8]
;; #[rbd–driver 40228d88])
; Remove the two line rubberband drivers.
(rbd:remove–type rbd:line?)
;; (#[rbd–driver 40228eb8] #[rbd–driver 40228d88])
; Remove the other rubberbander.
(rbd:clear #f)
;; ()
```

rbd:scheme

Scheme Extension: Rubberbanding

Action: Creates a user–defined rubberband driver and, optionally, makes it active.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: `(`**`rbd:scheme`**` activate hooks [locals])`

Arg Types:

activate	boolean
hooks	vector
locals	vector

Returns: rbd–driver

Errors: None

Description: hooks specifies a Scheme vector consisting of 7 elements. The elements are Scheme procedures. locals specifies a vector of user–defined objects. If activate is #t, this extension automatically adds the new driver to the list of active drivers; otherwise, this extension creates and returns it.

The Scheme procedures in the hooks vector are defined as follows:

```
hook[0] := (init–hook self)
```

is called once when the driver is activated. Can be used as globals or as elements in the locals vector.

```
hook[1] := (start–hook self pick–event)
```

is called when the driver is activated and when the mouse enters a window.

```
hook[2] := (update–hook self pick–event)
```

is called when the mouse moves within a window.

```
hook[3] := (stop–hook self)
```

is called when the mouse leaves a window and when the driver is deactivated.

```
hook[4] := (repaint–hook self view)
```

is called when a view receives a repaint event.

```
hook[5] := (position–hook self pick–event)
```

may be called by the other hooks to map a pick event to a position.

```
hook[6] := (end–hook self)
```

is called once when the driver is deactivated.

The argument types for the hooks are:

self is the rbd:scheme–driver	on whose behalf this hook is being called. This passes to all hooks so they can access any local variables in the locals vector.
pick–event	is the pick event that caused the hook to be called.
view	is the view in need of update by the hook called. The view can be obtained from pick–event as well.

The example below shows the hook procedure definition for a user–defined rubberband driver. It creates the seven–element Scheme vector, the–rb–hooks, whose procedures do nothing more than print out a simple message when an event occurs. Developers can replace these messages with more complex code to do application–specific tasks.

When the vector has been defined, a Scheme rubberband driver is created by calling rbd:scheme, using the the–rb–hooks vector. The #f tells rbd:scheme to create the driver but not activate it. The vector, the–locals, stores data specific to this particular driver. These data can be any Scheme objects and tailors the operation of a particular driver.

Finally, the rbd:push command pushes another other active rubberband driver list onto the stack and then makes this driver active at the top of the stack of active drivers. The rbd:pop command removes it from the top of the stack, deactivates it, and makes the pushed rubberband driver list on the stack the active driver. The read–event calls are a simple way to control the operation of the example.

The start position (when the cursor enters the window) is saved by the–start–hook in the driver's local variable vector(0). The end position is stored and modified by the–update–hook in the driver's local variable vector(1). These positions are printed when the example exits, to illustrate the use of local data.

Limitations: None

Example:

```
; rbd:scheme
; Create a vector to hold 7 elements.
; Define the rb hooks. This is called once when
; the driver is activated. It can be used to set
; globals or the elements in the locals vector.
(define the–rb–hooks (make–vector 7))
;; the–rb–hooks
; Define the init hook. This is Called when the
; driver is activated and when mouse enters a window.
(define the–init–hook (lambda (self)
    (display ”rubberband driver: INIT.\n”)))
;; the–init–hook
; Define the start hook. This is Called when the
; mouse moves within a window.
(define the–start–hook (lambda (self pick–event)
    (display ”rubberband driver: START.\n”)
    (rbd:scheme–set–local self 0
        (pick:position pick–event))
    (rbd:scheme–get–position self pick–event)))
;; the–start–hook
; Define the update hook. this is called when the
; mouse leaves a window and when the driver is
; deactivated.
(define the–update–hook (lambda (self pick–event)
    (display ”rubberband driver: UPDATE.\n”)
    (rbd:scheme–set–local self 1
        (pick:position pick–event))))
;; the–update–hook
; Define the stop hook. This is called when a view
; receives a repaint event.
(define the–stop–hook (lambda (self)
    (display ”rubberband driver: STOP.\n”)))
;; the–stop–hook
; Define the repaint hook. This may be called by
; other hooks to map a pick–event to a position.
(define the–repaint–hook (lambda (self view)
    (display ”rubberband driver: REPAINT.\n”)))
;; the–repaint–hook
; Define the position hook. This is Called once when
; the driver is deactivated.
(define the–position–hook (lambda (self pick–event)
    (display ”rubberband driver: POSITION.\n”)
    (display ”position not changed\n”)))
;; the–position–hook
; Define the end hook. Non–scheme rbd drivers can use
; the global variable ’rb–position–hook’, whereas,
; Scheme rbd drivers use that or the ’position
; hook’.
(define the–end–hook (lambda (self)
    (display ”rubberband driver: END.\n”)))
;; the–end–hook
; Initialize.
(vector–set! the–rb–hooks 0 the–init–hook)
;; ()
(vector–set! the–rb–hooks 1 the–start–hook)
;; ()
(vector–set! the–rb–hooks 2 the–update–hook)
;; ()
(vector–set! the–rb–hooks 3 the–stop–hook)
;; ()
(vector–set! the–rb–hooks 4 the–repaint–hook)
;; ()
(vector–set! the–rb–hooks 5 the–position–hook)
;; ()
(vector–set! the–rb–hooks 6 the–end–hook)
;; ()
; Start rubberbanding.
; Create and initialize some local variables.
(define the–locals (make–vector 2))
```

```
;; the–locals
(vector–set! the–locals 0 (position 0 0 0))
;; ()
(vector–set! the–locals 1 (position 0 0 0))
;; ()
; Create the Scheme rubberband driver.
(define the–scm–rbd
    (rbd:scheme #f the–rb–hooks the–locals))
;; the–scm–rbd
; Begin rubberbanding.
(read–event)
; Using the mouse button, select a screen
; position.
;; #[pick–event 185 455 1 1075924776 0]
; Push the rubberband driver onto the stack.
(rbd:push the–scm–rbd)
;; rubberband driver: INIT.
;; (#[rbd–driver 4021eb48])
(read–event)
;; rubberband driver: START.
;; rubberband driver: POSITION.
;; position not changed
;; rubberband driver: UPDATE.
;; ...
;; rubberband driver: UPDATE.
;; rubberband driver: STOP.
; Remove the driver from the stack.
(rbd:pop)
;; rubberband driver: END.
;; (#[rbd–driver 4021eb48])
; Display the results.
(display ”Start Position: ”)
    (display (vector–ref the–locals 0))
    (newline)
;; Start Position:
;; #[position 38.1656983627057 –92.3988426777293 0]
(display ”Stop Position: ”)
    (display (vector–ref the–locals 1))
    (newline)
;; Stop Position:
;; #[position 28.72230 –96.44886 –7.105e–15]
(rbd:clear #f)
;; ()
```

rbd:scheme–get–local

Scheme Extension: Rubberbanding

Action:	Gets a local variable in a Scheme rubberbander.
Filename:	gi/gi_scm/rb_scm.cxx
APIs:	None
Syntax:	`(`**`rbd:scheme-get-local`**` driver index)`

Arg Types:

driver	rbd–driver
index	integer

Returns: scheme–object

Errors: index is out of range.

Description: Scheme rubberbanders carry a vector of user–defined Scheme objects that are used within their hook functions. This extension enables the values in these objects to be obtained.

This extension is used by a Scheme rubberband driver to access the contents of its own “locals” vector.

Limitations: None

Example:

```
; rbd:scheme-get-local
; Define a start or update hook procedure that gets
; the value of the fourth local variable in a
; rbd:scheme driver’s locals vector.
(lambda (self event)
    (rbd:scheme-get-local self 3))
;; #[compound 1074571208]
```

rbd:scheme–get–position

Scheme Extension: Rubberbanding

Action:	Calls the local get position hook passing self and a pick event.
Filename:	gi/gi_scm/rb_scm.cxx
APIs:	None
Syntax:	`(`**`rbd:scheme-get-position`**` driver event)`

Arg Types:

driver	rbd–scheme–driver
event	pick–event

Returns: position | scheme–object

Errors: index is out of range

Description: Scheme rubberband drivers carry a vector of user–defined Scheme procedures. One of these is intended for calculating positions from an input pick event. This extension calls that procedure, passing a rbd–scheme–driver and a pick–event. It returns the value of the get position hook; normally this is a position, but it can be any scheme–object.

This extension is used by a Scheme rubberband driver to map a pick–event to a 3D position. This is achieved by calling rbd:scheme–get–position with the first argument as self.

Limitations: None

Example:
```
; rbd:scheme-get-position
; Define an update or start hook that calls the local
; Get the position hook of an rbd-scheme driver.
(lambda (self event)
   (rbd:scheme-get-position self event))
;; #[compound 1074571496]
```

rbd:scheme–set–local

Scheme Extension: Rubberbanding

Action: Sets a local variable in a Scheme rubberbander.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: (**rbd:scheme–set–local** driver index object)

Arg Types:

driver	rbd–driver
index	integer
object	scheme–object

Returns: scheme–object

Errors: index is out of range.

Description: Scheme rubberbanders carry a vector of user defined Scheme objects that are used within their hook functions. This extension enables new values to be set in these objects, and it returns the old value of the variable.

This extension is used by a Scheme rubberband driver to modify one of its own procedures. This is achieved by calling rbd:scheme–set–local with the first argument as self.

Limitations: None

Example:
```
; rbd:scheme-set-local
; Set the value of a local variable in a
; Scheme rubberband driver.
(lambda (self event)
    (rbd:scheme-set-local self 3 5))
;; #[compound 1074571992]
```

rbd:scheme?

Scheme Extension: Rubberbanding

Action: Determines if user–defined rubberband driver is in the active list of drivers.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: (**rbd:scheme?** object)

Arg Types:	object	scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if a user–defined rubberband driver is in the active list of rubberband drivers (e.g., at the top of the stack); otherwise, it returns #f.

Limitations: None

Example:
```
; rbd:scheme?
; Determine if there is an active Scheme rbd-driver.
(rbd:scheme? #t)
;; #f
```

rbd:start

Scheme Extension: Rubberbanding

Action: Sends the rubberbanding start event to all rubberbanders in the active list.

Filename:	gi/gi_scm/rb_scm.cxx	
APIs:	None	
Syntax:	(**rbd:start** event)	
Arg Types:	event	pick–event
Returns:	pick–event	
Errors:	None	

Description: If the rubberbanded figure is currently not displayed, this extension calls the rubberband driver's internal start–hook procedure for all rubberbanders in the active list. If the rubberbanded figure is currently displayed, this extension calls the rubberband driver's internal update–hook procedure for all rubberbanders in the active list.

event identifies the event used, typically, a mouse click to trigger the action.

This extension is used in conjunction with rbd:pop and rbd:push. It turns on the graphic portion of the rubberband drivers after it had been suspended (in the case of rbd:push and then rbd:pop).

Limitations: None

Example:

```
; rbd:start
; Send a rubberbanding start event.
(rbd:start (read–event))
; Using the mouse button, select a screen position.
;; #[pick–event 245 528 1 1075533160 0]
```

rbd:stop

Scheme Extension: Rubberbanding

Action:	Sends the rubberbanding stop event to all rubberbanders in the active list.
Filename:	gi/gi_scm/rb_scm.cxx
APIs:	None
Syntax:	(**rbd:stop**)
Arg Types:	None
Returns:	pick–event

Errors: None

Description: This extension calls the rubberband driver’s internal stop–hook procedure for all rubberbanders in the active list.

This extension is used in conjunction with rbd:pop and rbd:push. It turns off the graphic portion of the rubberband drivers so that (in the case of rbd:push) it will not be confused with the new driver being started.

Limitations: None

Example:

```
; rbd:stop
; Send a rubberbanding stop event.
(rbd:stop)
;; ()
```

rbd:update

Scheme Extension: Rubberbanding

Action: Sends the rubberbanding update event for all rubberbanders in the active list.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: (**rbd:update** event)

Arg Types: event pick–event

Returns: pick–event

Errors: None

Description: This extension calls the rubberband driver’s internal update–hook procedure for all rubberbanders in the active list.

event identifies the event used, typically, a mouse click to trigger the action.

This extension is used in conjunction with rbd:pop and rbd:push. It turns on the graphic portion of the rubberband drivers after it had been suspended (in the case of rbd:push and then rbd:pop).

Limitations: None

Example:

```
; rbd:update
; Send a rubberbanding update event.
(rbd:update (read–event))
;; #[pick–event 325 529 1 1075533160 0]
```

rbd:view

Scheme Extension: Rubberbanding, Viewing

Action: Creates a view rubberband driver and, optionally, makes it active.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax:

```
(rbd:view activate pe type [reverse=#t])
```

Arg Types:

activate	boolean
pe	pick–event
type	string
reverse	boolean

Returns: rbd–driver

Errors: None

Description: This extension rubber bands the view by moving the camera or moving the object based on the mouse position.

If activate is #t, this extension automatically adds the new driver to the list of active drivers; otherwise, this extension creates and returns it.

pe is a pick event used to control where rubber banding starts.

type is one of the following strings:

- trackball
- pan
- zoom
- orbit
- walk
- near_clip

If reverse is #t (the default), the motion like the object is being manipulated. If it is #f, then the motion is like the camera is being moved.

Limitations: None

Example:

```
; rbd:view
; Create an object to view the changes
(define block1
    (solid:block (position 0 0 0)
    (position 10 20 30)))
;; block1
; Create a view rubber band driver for rotating a
; view.
(rbd:view #t (read–event) ”trackball”)
;; #[rbd–driver 401b1df0]
; Click on a point in the view and move object
; Clear rubber band driver.
(rbd:clear #f)
;; ()


; A more detailed example from rotsph.scm.
; Run the following commands to create the
; rotate sphere rubber bander.
; To start the rotate sphere rubber bander,
; issue (rotsph #t). Drag the mouse while
; holding down the left button.
; To turn off, issue (rotsph #f).
(define rotsph
    (lambda (onoff)
        (begin
            (if onoff
; TRUE: Define mouse hooks to turn on and off
; rubber banding
    (begin
        ; When button pressed, start rubber banding
        (display ”Rubber band driver: DEFINE.\n”)
        (rbd:push)
        (define hhhrub #f)
        (display ”Hold mouse button ”)
        (display ”and drag to rotate camera.\n”)
        (set! mouse–down–hook
        (lambda (pe)
            (set! hhhrub
                (rbd:view #f pe ”orbit”))
            (dl:dynamic–silhouette–display #f)
            (rbd:add hhhrub)
            )
        )
        ; When button released, stop rubber banding
        (set! mouse–up–hook
                (lambda (pe)
                (rbd:remove hhhrub)
                (dl:dynamic-silhouette-display #t)
                )
            )
        )
; FALSE: Turn off rubber banding
        (begin
            ; Reset the mouse-hook operations
            (set! mouse-down-hook '())
            (set! mouse-up-hook '())
            (display "Rubber band driver: UNDEFINE.\n")
            (rbd:pop)
            (dl:dynamic-silhouette-display #t)
            )
        )
        ; Return the new setting
        onoff
        )
))
;; rotsph
```

rbd:window

Scheme Extension: Rubberbanding

Action:	Creates a window rubberband driver and, optionally, makes it active.
Filename:	gi/gi_scm/rb_scm.cxx
APIs:	None
Syntax:	(**rbd:window** activate base)
Arg Types:	activate boolean base pick–event
Returns:	rbd–driver
Errors:	None
Description:	base argument specifies a pick event used to define a corner of the window being rubberbanded. If activate is #t, this extension automatically adds the new driver to the list of active drivers; otherwise, this extension creates and returns it.

The window rubberband driver displays a white rectangle with one corner anchored at the pick event and the other end following the cursor. This extension is similar to rbd:rectangle, except that rubberbanding rectangle is always in the view plane instead of the the active WCS. It is intended for selecting a portion of the viewing area for zoom operations.

Limitations: None

Example:
```
; rbd:window
; Create an object to view the changes.
(define block1
    (solid:block (position 0 0 0)
    (position 10 20 30)))
;; block1
(define rubber-zoom
    (lambda ()
; Create first corner of view window
        (let ((corner1 (read-event)))
; Create a rbd-window driver, and make it active.
            (rbd:window #t corner1)
; Using the mouse, select second corner.
            (let ((corner2 (read-event)))
; stop the rubberbanding
            (rbd:remove-type rbd:window?)
; Use the rbd window as zoom window
            (view:zoom-window corner1 corner2)
; refresh the view so changes are visible
            (view:refresh)))))
;; rubber-zoom
; Issue the following command to recreate a
; zoom rubberbander. Follow the comments in
; the preceding define statement.
(rubber-zoom)
;; #[view 1075924776]
(rbd:clear #f)
;; ()
```

rbd:window?

Scheme Extension: Rubberbanding

Action: Determines if a window rubberband driver is in the active rubberband driver list.

Filename: gi/gi_scm/rb_scm.cxx

APIs: None

Syntax: (**rbd:window?** object)

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if the window rubberband driver is in the active rubberbander list; otherwise, it returns #f.

Limitations: None

Example:

```
; rbd:window?
; Determine if the Boolean is an rbd–window driver.
(rbd:window? #t)
;; #f
; Determine if the rbd–window driver is actually
; an rbd–window driver.
(rbd:window? (rbd:window #t (read–event)))
; Using the mouse, select a point.
;; #t
; Determine if the rbd–line driver is an rbd–window
; driver.
(rbd:window? (rbd:line #t (position 1 2 3)))
;; #f
; clear the rubberbanders
(rbd:clear #t)
;; ()
```

refresh–all

Scheme Extension: Viewing

Action: Refreshes all the views.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: (**refresh–all**)

Arg Types: None

Returns: string

Errors: None

Description: This extension refreshes all the views.

Limitations: None

Example:
```
; refresh–all
; Refresh the view.
(refresh–all)
;; ”refreshed”
```

render:rebuild

Scheme Extension: Rendering Control

Action: Rebuilds each part in the display list.

Filename: gi/gi_scm/dl_scm.cxx

APIs: api_gi_display_entity, api_gi_erase_entity

Syntax: (**render:rebuild** [part])

Arg Types:	part	part

Returns: unspecified

Errors: None

Description: This extension discards the existing entity display list and regenerates it from the entity data. This is useful when the view zooms in on entities, because the display may look jagged. This extension destroys the existing display list, regenerates it from the entities at the new zoom level, and redisplays the entities in all views associated with the part.

Limitations: None

Example:
```
; render:rebuild
; Rebuild and redisplay the display list.
(render:rebuild)
;; ()
; Refresh the display list for a view.
(view:refresh)
;; #[view 10755331601]
```

set–timer

Scheme Extension: Debugging

Action: Starts the timer or gets the elapsed time that the application has been running.

Filename:	gi/gi_scm/scm_timer.cxx	
APIs:	None	
Syntax:	(**set–timer** [on–off])	
Arg Types:	on–off	boolean
Returns:	string	
Errors:	None	
Description:	When set–timer is set to on (#t) this extension sets the application's timer to zero and returns the string "Timer Started". When set–timer is set to off (#f) the elapsed time since the application or timer was started is returned.	
Limitations:	None	

Example:

```
; set–timer
; Start the timer.
(set–timer #t)
;; "Timer Started"
; Get the elapsed time since the timer was started.
(set–timer #f)
;; "Time: 0.000000"
```