Chapter 4.
Scheme Extensions

Topic: Ignore

Scheme is a public domain programming language, based on the LISP language, that uses an interpreter to run commands. ACIS provides extensions (written in C++) to the native Scheme language that can be used by an application to interact with ACIS through its Scheme Interpreter. The C++ source files for ACIS Scheme extensions are provided with the product. *Spatial*'s Scheme based demonstration application, Scheme ACIS Interface Driver Extension (Scheme AIDE), also uses these Scheme extensions and the Scheme Interpreter. Refer to the *3D ACIS Online Help User's Guide* for a description of the fields in the reference template.

check–interrupt

Scheme Extension: Debugging, Picking

Action: Executes process_event to process pending events if interrupts are enabled.

Filename: scm/scmext/sys_scm.cxx

APIs: None

Syntax: (**check–interrupt**)

Arg Types: None

Returns: boolean

Errors: None

Description: This extension processes the pending messages used during long operations when the user wants to check for an interrupt condition. The returned value indicates the interrupt status.

Limitations: None

Example:

```
; check–interrupt
; Allow pending messages to be processed.
(check–interrupt)
;; #f
```

dl–item:color

Scheme Extension: Colors

Action: Gets the color of a display item in the display list.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: (**dl–item:color** item)

Arg Types: item dl–item

Returns: color

Errors: None

Description: Returns the color of the specified display item in the display list. A display item is not part of the model and does not get saved and restored.

item is a specified display item in the display list.

Limitations: None

Example:

```
; dl–item:color
; Define two polyline list items.
(define poly1 (dl–item:polyline
    (list (position 0 0 0) (position 5 0 0)
    (position 0 5 0) (position 0 0 0)) #t))
;; poly1
; Color poly1 red.
(dl–item:set–color poly1 1)
;; ()
(define poly2 (dl–item:polyline
    (list (position 10 10 10) (position 15 10 10)
    (position 10 15 10) (position 10 10 10)) #t))
;; poly2
; Color poly2 blue.
(dl–item:set–color poly2 3)
;; ()
; OUTPUT Display

; Request the color of poly1 and poly2.
(dl–item:color poly1)
;; #[color 1 0 0]
; color 1 0 0 is red.
(dl–item:color poly2)
;; #[color 0 0 1]
; color 0 0 1 is blue.
```

dl–item:display

Scheme Extension: Viewing

Action: Adds a display item to the display list for all views.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: (**dl–item:display** item)

Arg Types: item dl–item

Returns: unspecified

Errors: None

Description: This extension adds a display item to the display list for all views. This extension is used if the display list item is currently not being displayed, because it was turned off by the dl-item:erase extension. A display item is not part of the model and does not get saved and restored.

item is a specified display item in the display list.

Limitations: None

Example:

```
; dl–item:display
; Define two polyline list items.
(define poly1 (dl–item:polyline
    (list (position 0 0 0) (position 5 0 0)
    (position 0 5 0) (position 0 0 0)) #t))
;; poly1
; Color poly1 red.
(dl–item:set–color poly1 1)
;; ()
(define poly2 (dl–item:polyline
    (list (position 10 10 10) (position 15 10 10)
    (position 10 15 10) (position 10 10 10)) #t))
;; poly2
; Color poly2 blue.
(dl–item:set–color poly2 3)
;; ()
; Erase the first polyline.
(dl–item:erase poly1)
;; ()
; Re–display poly1 using featured command.
(dl–item:display poly1)
;; ()
```

dl–item:erase

Scheme Extension: Viewing

Action: Removes a display item from the display list.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: (**dl–item:erase** item)

Arg Types: item dl–item

Returns: unspecified

Errors: None

Description: This extension removes a display item from the display list in all views. view:refresh may be required to update the individual views. A display item is not part of the model and does not get saved and restored. This extension cannot be used for removing an entity from the display list; it only works on display items.

item is a specified display item in the display list.

Limitations: None

Example:

```
; dl–item:erase
; Define a point display list item.
(define db (dl–item:point (position 5 10 15)))
;; db
; Remove the point display list item.
(dl–item:erase db)
;; ()
(view:refresh)
;; #[view 1075640854]
```

dl–item:point

Scheme Extension: Viewing

Action: Creates a point item for the display list.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: (**dl–item:point** position)

Arg Types: position — position

Returns: dl-item

Errors: None

Description: This extension creates a point display list item at the specified position. The point is displayed in all views until it is deleted. The point style is controlled by env:set-point-style and env:set-point-size.

A display item is not part of the model and does not get saved and restored. Even though a 3D position is provided for the display item, it is mapped into 2D space. A display item should then not be used for snapping or as part of a model.

position is where a point display item is created.

Limitations: None

Example:

```
; dl–item:point
; Create a point display list item.
(define dp (dl–item:point (position 15 5 10)))
;; dp
```

dl–item:polyline

Scheme Extension: Viewing

Action: Creates a polyline item for the display list.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: `(dl–item:polyline position–list [fill=#f])`

Arg Types:

position–list	position | (position ...)
fill	boolean

Returns: dl-item

Errors: None

Description: This extension creates a polyline display item for the display list connecting the specified positions. The optional fill flag indicates whether the polyline is filled. The polyline is displayed in all views until it is removed.

A display item is not part of the model and does not get saved and restored. Even though a 3D position is provided for the display item, it is mapped into 2D space. A display item should then not be used for snapping or as part of a model.

position–list gives the list of positions.

The optional fill flag indicates whether the polyline is filled.

Limitations: None

Example:
```
; dl–item:polyline
; Create a polyline display list item
; for a filled triangle.
(define poly (dl–item:polyline (list
    (position 0 0 0) (position 5 0 0)
    (position 0 5 0) (position 0 0 0)) #t))
;; poly
```

dl–item:set–color

Scheme Extension: Colors

Action: Sets the color of a display item.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: (**dl–item:set–color** item color)

Arg Types:		
	item	dl–item
	color	color

Returns: unspecified

Errors: None

Description: Sets the color of the specified display item. A display item is not part of the model and does not get saved and restored.

Limitations: None

Example:
```
; dl–item:set–color
; Create a text display list item.
(define dt (dl–item:text (position 0 0 25) ”Note”))
;; dt
; Set the display color of the text
; display list item to red.
(dl–item:set–color dt (color:rgb 1 0 0))
;; ()
```

dl–item:text

Scheme Extension: Text

Action: Creates a text display list item.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: `(dl–item:text position string)`

Arg Types:

position	position
string	string

Returns: dl-item

Errors: None

Description: This extension creates a text display list item at the specified position. The text displays in all views until it is removed.

A display item is not part of the model and does not get saved and restored. Even though a 3D position is provided for the display item, it is mapped into 2D space. A display item should then not be used for snapping or as part of a model.

position is where a text display list item is created.

string is to be displayed.

Limitations: None

Example:

```
; dl–item:text
; Create a text display list item.
(define dt (dl–item:text (position 10 0 15) ”Hello”))
;; dt
; Create a second text display list item.
(define ds (dl–item:text (position –10 0 15) ”Test”))
;; ds
(dl–item:set–color dt (color:rgb 1 0 0))
;; ()
```

dl–item?

Scheme Extension: Viewing

Action: Determines if a Scheme object is a display item.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: `(dl–item? object)`

Arg Types:	object	scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if the specified object is a display list item; otherwise, it returns #f. A display item is not part of the model and does not get saved and restored.

object is a specified Scheme object of a display list item.

Limitations: None

Example:

```
; dl–item?
; Create a point display list item.
(define d1 (dl–item:point (position 0 0 0)))
;; d1
; Verify that the object is a display list item.
(dl–item? d1)
;; #t
; Create a circular edge.
(define edge1 (edge:circular (position 0 0 0) 20))
;; edge1
; Verify that the circular edge is not
; a display list item.
(dl–item? edge1)
;; #f
```

edge:chain

Scheme Extension: Model Topology

Action: Gets a chain of edges given one or two edges.

Filename: scm/scmext/pick_scm.cxx”

APIs: None

Syntax: `(edge:chain start–edge–entity [stop–edge–entity])`

Arg Types:	start–edge–entity	edge
	stop–edge–entity	edge

Returns: (entity ...)

Errors: None

Description: Returns a chain of edges given one or two edges. This extension operates on top level edges only.

The argument start–edge–entity specifies an edge entity for the start.

The optional argument stop–edge–entity specifies an edge entity for termination.

Note *The edges of the chain must link to one another through common end points.*

Limitations: None

Example:

```
; edge:chain
; Create linear edge 1.
(define edge1
    (edge:linear (position 5 5 0) (position 20 5 0)))
;; edge1
; Create linear edge 2.
(define edge2 (edge:linear (position 18 0 0)
    (position 18 20 0)))
;; edge2
; Create linear edge 3.
(define edge3
    (edge:linear (position 18 0 0)
    (position 15 15 0)))
;; edge3
(define edge4
    (edge:linear (position 15 15 0)
    (position 10 9 0)))
;; edge4
; Determine which edge forms a chain.
(edge:chain edge1)
;; ()
; Determine which edge forms a chain.
(edge:chain edge2 edge3)
;; (#[entity 3 1] #[entity 4 1])
(edge:chain edge2 edge4)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1])
```

entity:color

Scheme Extension: Entity, Colors

Action: Gets the display color for an entity.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: `(entity:color entity)`

Arg Types:	entity	entity

Returns: color

Errors: None

Description: This extension returns a color object.

entity is a Scheme entity for a color object.

Limitations: None

Example:

```
; entity:color
; Create entity 1.
(define edge1
    (edge:circular (position 0 0 0) 25 0 180))
;; edge1
; Create entity 2.
(define edge2
    (edge:circular (position 5 5 5) 25 180 270))
;; edge2
; Create entity 3.
(define edge3
    (edge:linear (position 0 0 0) (position 25 0 0)))
;; edge3
; Create entity 4.
(define block1
    (solid:block (position 0 0 0)
    (position 15 15 15)))
;; block1
; Set the display color for entities 1 and 3.
(entity:set–color (list edge1 edge3) 5)
;; ()
; Set the display color for entity 2.
(entity:set–color edge2 1)
;; ()
; Get the display color for entity 1.
(entity:color edge1)
;; #[color 1 1 0]
; Get the display color for entity 2.
(entity:color edge2)
;; #[color 1 0 0]
```

entity:display

Scheme Extension: Entity, Viewing

Action: Displays entities.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: (**entity:display** entity–list)

Arg Types: entity–list entity | (entity ...)

Returns: (entity ...)

Errors: None

Description: The argument entity–list specifies an entity or list of entities to display. This extension returns the input entity–list.

Use this extension to display entities that were not displayed when they were created, or to redisplay entities that have been erased. The display for the entities always recomputes, so use it to regenerate the display of an entity with a different faceting tolerance if the view scale has changed and the display is now too coarse.

Limitations: None

Example:

```
; entity:display
; Create an edge.
(define edge1
    (edge:circular (position 0 0 0) 25 0 180))
;; edge1
; Turn off the automatic display of new entities.
(env:set–auto–display #f)
;; ()
; Create a second edge.
(define edge2
    (edge:circular (position 5 5 5) 25 180 270))
;; edge2
; Force the display of entity 2.
(define display (entity:display edge2))
;; display
```

entity:displayed?

Scheme Extension: Entity, Viewing

Action: Determines if a Scheme entity is displayed.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: (**entity:displayed?** entity)

Arg Types:	entity	entity

Returns: boolean

Errors: None

Description: Refer to Action.

entity is an input Scheme entity.

Limitations: None

Example:
```
; entity:displayed?
; Create a block.
(define block1
    (solid:block
    (position 0 0 0) (position 6 6 6)))
;; block1
; Determine if the block1 is displayed.
(entity:displayed? block1)
;; #t
; Erase the block.
(define erase (entity:erase block1))
;; erase
; Determine if the block is displayed.
(entity:displayed? block1)
;; #f
```

entity:erase

Scheme Extension: Entity

Action: Erases but does not remove the specified entity or list of entities.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: (**entity:erase** entity–list)

Arg Types:	entity–list	entity | (entity ...)

Returns: (entity ...)

Errors: None

Description: The argument entity–list is an entity or list of entities to be erased from the display. The entities remain on hand for later redisplay. To redisplay an erased entity or list of entities, use the entity:display extension. This extension returns the input entity–list.

Limitations: None

Example:

```
; entity:erase
; Create an edge.
(define edge1
    (edge:circular (position 0 0 0) 25 0 180))
;; edge1
; Create another edge.
(define edge2
    (edge:circular (position 5 5 5) 25 180 270))
;; edge2
; Erase the second edge.
(define erase (entity:erase edge2))
;; erase
; Display the erased entity 2.
(define display (entity:display edge2))
;; display
; The erased entity is redisplayed.
```

entity:highlighted?

Scheme Extension: Entity, Highlighting

Action: Determines if an entity is highlighted.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: (**entity:highlighted?** entity)

Arg Types:	entity	entity

Returns: boolean

Errors: None

Description: Refer to Action.

entity is an input Scheme entity.

Limitations: None

Example:

```
; entity:highlighted?
; Create circular edge 1.
(define edge1
    (edge:circular (position 0 0 0) 25 0 180))
;; edge1
; Create circular edge 2.
(define edge2
    (edge:circular (position 5 5 5) 25 180 270))
;; edge2
; Highlight circular edges 1 and 2.
(define highlight (entity:set–highlight
    (list edge1 edge2) #t))
;; highlight
; Determine if circular edge 1 is highlighted.
(entity:highlighted? edge1)
;; #t
; Determine if circular edge 2 is highlighted.
(entity:highlighted? edge2)
;; #t
```

entity:part

Scheme Extension: Entity, Physical Properties

Action: Gets the part to which an entity belongs.

Filename: scm/scmext/eprt_scm.cxx

APIs: None

Syntax: (**entity:part** entity)

Arg Types: entity entity

Returns: part

Errors: None

Description: This extension returns the part to which the entity belongs. If the entity does not belong to any part, this extension returns #f.

entity is an input Scheme entity.

Limitations: None

Example:

```
; entity:part
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 30 30 30)))
;; block1
; Determine to which part the block belongs.
(entity:part block1)
;; #[part 1]
```

entity:select

Scheme Extension: Entity

Action: Selects an entity or list of entities.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: (**entity:select** entity–list {on–off})

Arg Types:		
	entity–list	entity | (entity ...)
	on–off	boolean

Returns: (entity ...)

Errors: None

Description: The argument entity-list specifies an entity or list of entities to be selected or deselected. This extension returns the input entity-list. The initialized selection color is red. Individual faces may also be selected.

The argument on selects the specified entities.

The argument off deselects the specified entities.

Note To display selection, change the default color of the selection so that it is different from the color of the entity.

Limitations: None

Example:

```
; entity:select
; Create circular edge 1.
(define edge1
    (edge:circular (position 0 0 0) 25 0 180))
;; edge1
; Create circular edge 2.
(define edge2
    (edge:circular (position 5 5 5) 25 180 270))
;; edge2
; Select the first edge.
(entity:select edge1 #t)
;; #[entity 2 1]
; Select entities 1 and 2.
(entity:select (list edge1 edge2) #t)
;; (#[entity 2 1] #[entity 3 1])
```

entity:selected?

Scheme Extension: Entity

Action: Determines if an entity is selected.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: (**entity-selected?** entity)

Arg Types:	entity	entity

Returns: boolean

Errors: None

Description: Refer to Action.

entity is an input Scheme entity.

Limitations: None

Example:

```
; entity:selected?
; Create circular edge 1.
(define edge1
    (edge:circular (position 0 0 0) 25 0 180))
;; edge1
; Create circular edge 2.
(define edge2
    (edge:circular (position 5 5 5) 25 180 270))
;; edge2
; Select circular edges 1 and 2.
(define selected (entity:select
    (list edge1 edge2) #t))
;; select
; Determine if circular edge 1 is selected.
(entity:selected? edge1)
;; #t
; Determine if circular edge 2 is selected.
(entity:selected? edge2)
;; #t
```

entity:set–color

Scheme Extension: Entity, Colors

Action: Sets the display color for an entity or list of entities.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: (**entity:set–color** entity–list color)

Arg Types:		
	entity–list	entity | (entity ...)
	color	color

Returns: color

Errors: None

Description: The argument entity–list specifies an entity or list of entities to be assigned a color.

The argument color can accept an integer or a color:rgb value. color specifies a new color to assign to specified entities; color values include:

Black = 0 . . . . . = #[color 0 0 0]
Red = 1 . . . . . = #[color 1 0 0]
Green = 2 . . . . . = #[color 0 1 0]
Blue =3 . . . . . = #[color 0 0 1]
Cyan = 4 . . . . . = #[color 0 1 1]
Yellow = 5 . . . . . = #[color 1 1 0]
Magenta = 6 . . . . . = #[color 1 0 1]
White = 7 . . . . . = #[color 1 1 1]

This extension returns the entity's previous color.

Limitations: None

Example:

```
; entity:set–color
; Create a solid block.
(define block1
    (solid:block (position 0 –5 0)
    (position 15 15 15)))
;; block1
; OUTPUT Original

; Set the color for the block to red
(entity:set–color block1 1)
;; ()
; OUTPUT Result
```

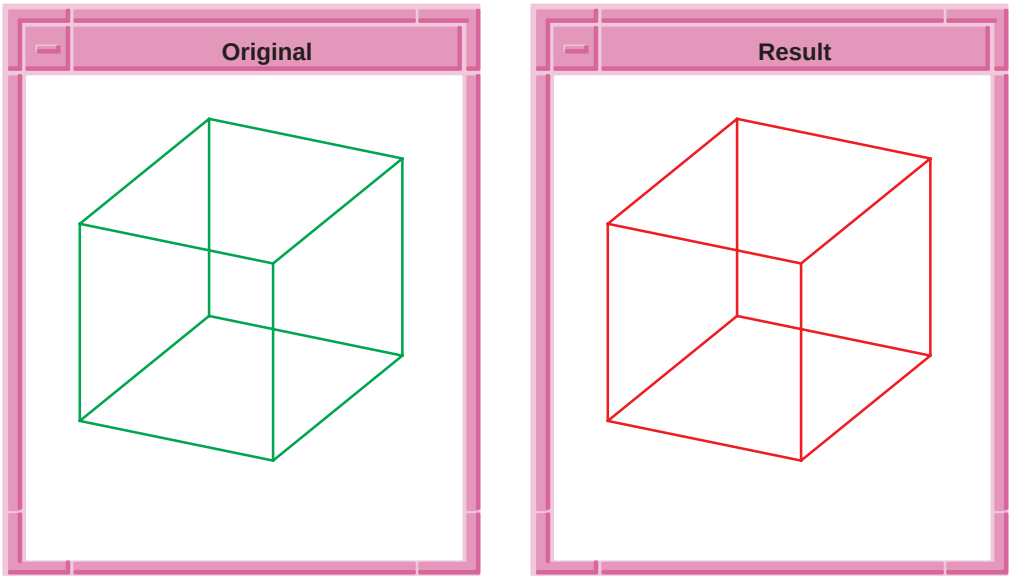

Figure 4-1. entity:set–color

entity:set–highlight

Scheme Extension: Entity, Highlighting

Action: Sets highlighting for an entity or list of entities.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: `(entity:set–highlight entity–list {on–off})`

Arg Types:		
	entity–list	entity | (entity ...)
	on–off	boolean

Returns: (entity ...)

Errors: None

Description: The argument entity–list specifies an entity or list of entities to be highlighted or unhighlighted. This extension returns the input entity–list.

on activates highlighting for the specified entities.

The argument off deactivates highlighting for the specified entities. The initialized highlight color is white. Individual faces may also be highlighted.

Note *To display highlighting, change the default color of the highlight so that it is different from the color of the entity.*

Limitations: None

Example:

```
; entity:set–highlight
; Create circular edge 1.
(define edge1
    (edge:circular (position 0 0 0) 25 0 180))
;; edge1
; Create circular edge 2.
(define edge2
    (edge:circular (position 5 5 5) 25 180 270))
;; edge2
; Set the highlight for the first edge to on.
(entity:set–highlight edge1 #t)
;; #[entity 2 1]
; Set the highlight for entities 1 and 2 to on.
(entity:set–highlight (list edge1 edge2) #t)
;; (#[entity 2 1] #[entity 3 1])
```

entity:set–part

Scheme Extension: Entity, Part Management

Action: Sets the part to which an entity belongs.

Filename: scm/scmext/eprt_scm.cxx

APIs: api_pm_add_entity

Syntax: (**entity:set–part** entity [part=active])

Arg Types:		
	entity	entity
	part	part

Returns: entity

Errors: None

Description: If the entity already belongs to a different part, it is removed from the original part, and added to the new part.

Use this extension to move entities from one part to another. The argument entity specifies the entity to move from the original part. The argument part specifies the part to move the entity to. If it already belongs to the given part or active part, it remains unchanged. If it belongs to another part other than the destination part, a new entity is returned having all of the properties of the original. The original entity is deleted from its owning part and is moved to the destination part. Because the original entity is deleted, it is wise to capture the new entity in a variable so that it can be referred to.

Limitations: If entity is not a top level entity, it cannot be added to a part that is different from its owner's.

Example:

```
; entity:set–part
; Create an active view for display.
; This view will display the default or part 1.
(define view1 (view:dl 1 1 256 256))
;; view1
; Create a part
(define first–p (part:new))
;; first–p
; Create a 2nd part.
(define second–p (part:new))
;; second–p
; Create a view to display the 2nd part.
(define view2 (view:dl 1 1 256 256 second–p))
;; view2
; Make sure the active part is part 1 or default.
(env:active–part)
;; #[part 1]
; Create a block in the active part.
(define block1
    (solid:block (position 0 0 0)
    (position 10 20 30)))
;; block1
; Get the ID of the block.
(entity:part block1)
;; #[part 1]
; Move the block from part 1 to part 2.
(define move (entity:set–part block1 second–p))
;; move
; Distributes entities to stream it
; was not created in.
(define new–ent (car (part:entities second–p)))
;; new–ent
(define new (entity:part new–ent))
;; new
```

entity:set–render

Scheme Extension: Entity, Rendering Control

Action: Marks whether or not an entity is to be rendered.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: (**entity:set–render** entity–list on–off=#t)

Arg Types:	entity–list	entity | (entity ...)
	on–off	boolean

Returns: (entity ...)

Errors: None

Description: entity–list specifies an entity or list of entities to be rendered or not. This extension returns the input entity–list.

If on–off is #t, it means that the entity will be rendered; if on–off is #f, it won't be rendered. The default state for entities is #t.

Limitations: The effect of this won't be seen until an entity:display is performed on the entities in the list.

Example:

```
; entity:set–render
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 30 10 20)))
;; block1
; Create a solid block.
(define block2
    (solid:block (position 30 10 20)
    (position 45 40 25)))
;; block2
; Render both blocks.
(render)
;; ()
; Set rendering off for one of the blocks.
(define set–render (entity:set–render block1 #f))
;; set–render
(define display (entity:display block1))
;; display
```

entray:position

Scheme Extension: Ray Testing

Action: Gets the position of the vertex of the entity component of an entray that is closest to the ray.

Filename: scm/scmext/qray_scm.cxx

APIs: None

Syntax: `(entray:position entray)`

Arg Types: entray entray

Returns: position

Errors: None

Description: Gets the position of the vertex of the entity component of an entray that is closest to the ray.

An entray is a Scheme object with an entity component and a ray component. Entrays are represented externally in the form #[entray n p (x y z) (a b c)], indicating its entity components are #[entity n p] and its ray components are #[ray (x y z) (a b c)].

Limitations: None

Example:

```
; entray:position
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 15 15 15)))
;; block1
; Create a entity ray and then extract
; the position portion.
(entray:position (entray block1
    (ray (position 0 0 0) (gvector 0 0 1))))
;; #[position 0 0 15]
```

entray:vertex

Scheme Extension: Ray Testing

Action: Gets the vertex on the entity component of the entray that is closest to the ray.

Filename: scm/scmext/qray_scm.cxx

APIs: None

Syntax: `(entray:vertex entray)`

Arg Types: entray entray

Returns: vertex

Errors: None

Description: Gets the vertex on the entity component of the entray that is closest to the ray.

An entray is a Scheme object with an entity component and a ray component. Entrays are represented externally in the form #[entray n p (x y z) (a b c)], indicating its entity components are #[entity n p] and its ray components are #[ray (x y z) (a b c)].

Limitations: None

Example:

```
; entray:vertex
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 15 15 15)))
;; block1
; Create an entity ray consisting of
; the block and a ray, and then find
; the vertex of the entity closest to the ray.
(entray:vertex (entray block1
    (ray (position 0 0 0) (gvector 0 0 1))))
;; #[entity 3 1]
```

env:active–part

Scheme Extension: Part Management, Physical Properties

Action: Gets the active part.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: (**env:active–part**)

Arg Types: None

Returns: part

Errors: None

Description: Gets the active part. A part is a grouping of top–level entities. When entities are created, they are added to the active part, unless specified otherwise. A part is represented externally in the form #[part n], where *n* represents its identification number.

active–part is a part that is a grouping of top–level activities.

Limitations: None

Example:

```
; env:active–part
; Get the ID of the currently active part.
(env:active–part)
;; #[part 1]
; Create another part.
(define part1 (part:new))
;; part1
; Verify what was created.
part1
;; #[part 2]
; Get the number of parts currently defined.
(env:count–parts)
;; 2
; Change the active part to the new one.
(env:set–active–part part1)
;; ()
; Verify that the new part is the active part.
(env:active–part)
;; #[part 2]
; Anything created from this point in time
; is added to the second part.
; Change the active part to the old one.
(env:set–active–part (part 1))
;; ()
; Verify that the old part is now the active part.
(env:active–part)
;; #[part 1]
```

env:active–view

Scheme Extension: Scheme Interface, Viewing

Action: Gets the currently-active view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**env:active–view**)

Arg Types: None

Returns: view

Errors: None

Description: Gets the active view. A view is assigned to an individual part. Only the entities in that part are displayed. When entities are created, they are added to the active part. When views are created, they display the entities from the active part, unless specified otherwise. A view is represented externally in the form #[view n], where n represents its identification number. Multiple views are helpful, for example, for displaying wireframe and rendered representations of the same part at the same time in two different windows. Setting the active view is useful for operations such as the render Scheme extension, because they assume the active view unless specified otherwise. Separate views can be used to display separate parts or the same part. When separate views are assigned to the same part, they can have different eye and target positions which present different views.

active–view is a view that is assigned to an individual part.

Limitations: None

Example:

```
; env:active–view
; Create a new view.
(define view1 (view:gl))
;; view1
; Create a new view.
(define view2 (view:gl))
;; view2
; Verify ID of the views.
view1
;; #[view 3867314]
view2
;; #[view 268829534]
; Get the active view.
; Define the active view.
(env:set–active–view view2)
;; ()
; verify active view.
(env:active–view)
;; #[view 268829534]
; Change the active view.
(env:set–active–view view1)
;; ()
; Verify the new active view.
(env:active–view)
;; #[view 3867314]
```

env:active–wcs–color

Scheme Extension: Colors, Work Coordinate Systems

Action: Gets the display color of the active WCS.

Filename: scm/scmext/env_scm.cxx

APIs: None

Syntax: (**env:active–wcs–color**)

Arg Types: None

Returns: color

Errors: None

Description: Gets the display color of the active WCS. This is an RGB color used to display the active WCS. By default, this value is #[color 0 1 0].

active–wcs–color **gives the color of the active WCS.**

Limitations: None

Example:

```
; env:active–wcs–color
; Determine if there is an active WCS.
(wcs:active)
;; #f
; Define a WCS.
(define wcs
    (wcs (position 0 0 0) (position 5 0 0)
    (position 0 5 0)))
;; wcs
; Make a WCS active.
(wcs:set–active wcs)
;; ()
; Get the color of the active WCS.
(env:active–wcs–color)
;; #[color 0 1 0]
; Set the color of the active WCS.
(env:set–active–wcs–color 6)
;; ()
; Get the color of the active WCS.
(env:active–wcs–color)
;; #[color 1 0 1]
```

env:auto–display

Scheme Extension: Viewing, Physical Properties

Action: Gets the auto–display mode.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: (**env:auto–display**)

Arg Types: None

Returns: boolean

Errors: None

Description: Gets the auto–display mode. The automatic display at #t shows geometry entities in all views as the entities are created. If set to #f, geometry entities are displayed only in the active view as they are created. When deactivated, new entities are only created in the active view. An example of where this is useful is when rendered and wireframe views of the same part are used at the same time. The default for auto–display is #t. To change the display state of previously created entities, use the extensions entity:erase and entity:display for those entities.

auto–display is a environment that has a on and off mode.

Limitations: None

Example:

```
; env:auto–display
; Determine whether the environment auto display
; of entities is on or off.
(env:auto–display)
;; #t
; Disable the automatic display of
; newly created entities.
(env:set–auto–display #f)
;; ()
; Verify that auto–display is disabled.
(env:auto–display)
;; #f
; Enable the automatic display of
; newly created entities.
(env:set–auto–display #t)
;; ()
; Verify that auto–display is enabled.
(env:auto–display)
;; #t
```

env:count–parts

Scheme Extension: Part Management, Physical Properties

Action: Gets the number of defined parts.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: (**env:count–parts**)

Arg Types: None

Returns: integer

Errors: None

Description: Refer to Action.

count–part is an enviroment that counts and returns the number of parts.

Limitations: None

Example:

```
; env:count–parts
; Define an active part.
(define first (env:active–part))
;; first
; Create another part.
(define part1 (part:new))
;; part1
; part1
; Verify what was created.
part1
;; #[part 2]
; Get the number of parts currently defined.
(env:count–parts)
;; 2
```

env:default–color

Scheme Extension: Colors, Scheme Interface

Action: Gets the current default color value setting.

Filename: scm/scmext/env_scm.cxx

APIs: None

Syntax: (**env:default–color**)

Arg Types: None

Returns: color

Errors: None

Description: Gets the RGB display color for newly created entities. By default, this value is #[color 0 1 0]. For the results to be visible, the entity should be reset with entity:set–color.

default–color displays the RGB color for newly created entities. The default value is #[color 0 1 0].

Limitations: None

Example:

```
; env:default–color
; Create a block.
(define block1
    (solid:block (position 12 0 5)
    (position 24 10 –8)))
;; block1
; Get the default color as an rgb value.
(env:default–color)
;; #[color 0 1 0]
; Set the default color for newly created entities
; to cyan using an integer.
(env:set–default–color 4)
;; ()
; Get the default color as an rgb value.
(env:default–color)
;; #[color 0 1 1]
; Create a sphere using the new default color.
(define sphere1
    (solid:sphere (position 4 4 4) 8))
;; sphere1
; Set a new default color as an rgb value.
(env:set–default–color (color:rgb 0.8 0.5 0.2))
;; ()
; Get the default color as an rgb value.
(env:default–color)
;; #[color 0.8 0.5 0.2]
; View the first entity the new default color.
(entity:set–color block1 (env:default–color))
;; ()
```

env:highlight–color

Scheme Extension: Colors, Highlighting

Action: Gets the highlight color.

Filename: scm/scmext/env_scm.cxx

APIs: None

Syntax: (**env:highlight–color**)

Arg Types: None

Returns: color

Errors: None

Description: This extension returns the current highlight color RGB value. By default, this value is #[color 1 1 1]. The highlight color setting affects all views. A change in highlight color affects entities that have highlighting enabled. Use the Scheme extension entity:set–highlight to enable highlighting for an entity.

highlight–color contains the highlight RGB color. The default value is #[color 1 1 1].

Limitations: None

Example:

```
; env:highlight–color
; Create a block.
(define block1
    (solid:block (position 12 0 5)
    (position 24 10 –8)))
;; block1
; OUTPUT Original

; Get the highlight color as an rgb value.
(env:highlight–color)
;; #[color 1 1 1]
; Set the display color for highlighted entities
; as an integer.
(env:set–highlight–color 3)
;; ()
; Get the highlight color as an rgb value.
(env:highlight–color)
;; #[color 0 0 1]
; Define a list of faces for the block.
(define faces1 (entity:faces block1))
;; faces1
; Highlight the first face in the list.
(define highlight (entity:set–highlight
    (car faces1) #t))
;; highlight
; OUTPUT Result
```

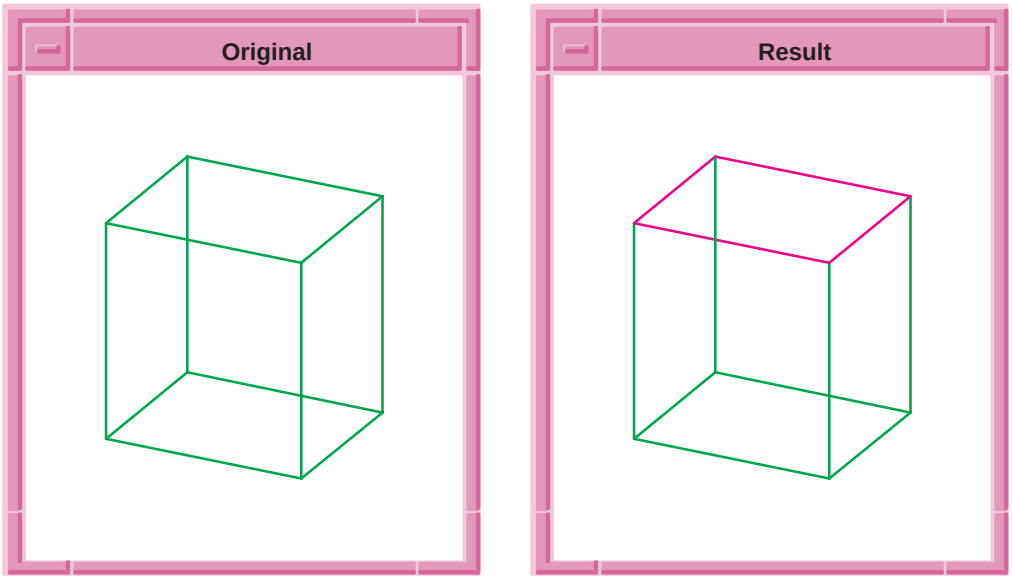

Figure 4-2. env:highlight–color

env:parts

Scheme Extension: Part Management, Physical Properties

Action: Gets a list of all parts in the environment.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: **(env:parts)**

Arg Types: None

Returns: (part ...)

Errors: None

Description: This extension returns a list of all parts in the environment.

parts is an environment that is a grouping of entities.

Limitations: None

Example:

```
; env:parts
; Create a part.
(define first (env:active–part))
;; first
; Create another part.
(define part1 (part:new))
;; part1
; part1
; Get the number of parts currently defined.
(env:count–parts)
;; 2
; Get a list of all parts in the environment.
(env:parts)
;; (#[part 1] #[part 2])
```

env:point–size

Scheme Extension: Viewing, Scheme Interface

Action: Gets the display size of a point.

Filename: scm/scmext/env_scm.cxx

APIs: None

Syntax: (**env:point–size**)

Arg Types: None

Returns: integer

Errors: None

Description: This extension returns the current display size of a point. This pertains to points displayed as locations, and does not pertain to the text font, point size.

point–size gives the display size of the point.

Limitations: None

Example:

```
; env:point–size
; Create a point.
(define point–position (point (position 5 4 10)))
;; point–position
(env:point–size)
;; 10
; Set the display size for a point.
(env:set–point–size 20)
;; ()
; Get the display size for a point.
(env:point–size)
;; 20
```

env:point–style

Scheme Extension: Viewing, Scheme Interface

Action: Gets the display style of a point.

Filename: scm/scmext/env_scm.cxx

APIs: None

Syntax: (**env:point–style**)

Arg Types: None

Returns: string

Errors: None

Description: This extension returns the point style as ”+”, ”X”, ”.” (period), or ”O” (the letter ”O”) displayed in the view as a square.

point–style is the current style of the point.

Limitations: None

Example:

```
; env:point-style
; Set the display style for a point to a +.
(env:set-point-style "+")
;; ()
; Get the display style for a point.
(env:point-style)
;; "+"
; Create a point.
(define point-position (point (position -5 -5 -5)))
;; point-position
; Set the display style of a point to a square.
(env:set-point-style "o")
;; ()
; Create another point.
(define point2 (point (position 5 5 5)))
;; point2
(env:point-style)
;; "O"
```

env:set–active–part

Scheme Extension: Part Management

Action: Sets the active part.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: (**env:set-active-part** part)

Arg Types:	part	part

Returns: unspecified

Errors: None

Description: Sets the active part. A part is a container for entities. When entities are created, they are added to the active part. A part is represented externally in the form #[part n], where *n* represents its identification number. Only one part can be assigned to any given view. However, multiple views can display the entities from the same part.

part is an environment that is a grouping of entities.

Limitations: None

Example:

```
; env:set–active–part
; Get the ID of the currently active part.
(env:active–part)
;; #[part 1]
; Create another part.
(define part1 (part:new))
;; part1
; Verify what was created.
part1
;; #[part 2]
; Change the active part to the new one.
(env:set–active–part part1)
;; ()
; Verify the active part.
(env:active–part)
;; #[part 2]
; Anything created from this point in time
; is added to the second part.
; Change the active part to the old one.
(env:set–active–part (part 1))
;; ()
; Verify the active part.
(env:active–part)
;; #[part 1]
```

env:set–active–view

Scheme Extension: Viewing, Scheme Interface

Action: Sets the specified view as the active view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**env:set–active–view** view)

Arg Types:	view	view

Returns: unspecified

Errors: None

Description: Sets the active view. A view is assigned to an individual part. Only the entities in that part are displayed. When entities are created, they are added to the active part. When views are created, they display the entities from the active part, unless specified otherwise. A view is represented externally in the form #[view n], where *n* represents its identification number. Multiple views are helpful, for example, for displaying wireframe and rendered representations of the same part at the same time in two different windows. Setting the active view is useful for operations such as the render Scheme extension, because they assume the active view unless specified otherwise. Separate views can be used to display separate parts or the same part. When separate views are assigned to the same part, they can have different eye and target positions which presents different views.

view is assigned to an individual part.

Limitations: None

Example:

```
; env:set–active–view
; Create view 1.
(define view1 (view:dl))
;; view1
; Create view 2.
(define view2 (view:dl))
;; view2
; Verify the views.
view1
;; #[view 10755193761]
view2
;; #[view 10755193798]
; Get the ID of the new (active) view.
(env:active–view)
;; #[view 10755193761]
; Change the active view.
(env:set–active–view view1)
;; ()
; Get the ID of the new (active) view.
(env:active–view)
;; #[view 10755193761]
```

env:set–active–wcs–color

Scheme Extension: Colors, Work Coordinate Systems

Action: Sets the color of the active WCS.

Filename: scm/scmext/env_scm.cxx

APIs: None

Syntax: (**env:set–active–wcs–color** color=0 1 0)

Arg Types: color color

Returns: unspecified

Errors: None

Description: Sets the display color of the active WCS. This is an RGB color. It is the color used to display the active WCS. By default, this value is #[color 0 1 0]. In order for the results to be visible, the active WCS must be set with wcs:set–active.

color is one of the RGB colors. The valid colors being:

Black = 0 . . . . . = #[color 0 0 0]
Red = 1 . . . . . = #[color 1 0 0]
Green = 2 . . . . . = #[color 0 1 0]
Blue =3 . . . . . = #[color 0 0 1]
Cyan = 4 . . . . . = #[color 0 1 1]
Yellow = 5 . . . . . = #[color 1 1 0]
Magenta = 6 . . . . . = #[color 1 0 1]
White = 7 . . . . . = #[color 1 1 1]

Limitations: None

Example:

```
; env:set–active–wcs–color
; Determine if there is an active WCS.
(wcs:active)
;; #f
; If there is no active WCS, define a WCS.
(define wcs
    (wcs (position 0 0 0) (position 5 0 0)
    (position 0 5 0)))
;; wcs
; Make the WCS active.
(wcs:set–active wcs)
;; ()
; Get the color of the active WCS.
(env:active–wcs–color)
;; #[color 0 1 0]
; Change the color of the active WCS.
(env:set–active–wcs–color 6)
;; ()
; Get the color of the active WCS.
(env:active–wcs–color)
;; #[color 1 0 1]
```

env:set–auto–display

Scheme Extension: Viewing

Action: Sets the automatic display of newly–created entities on or off.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: `(env:set–auto–display on–off=#t)`

Arg Types:

on–off	boolean

Returns: unspecified

Errors: None

Description: The argument on–off activates the automatic display in all views of geometry entities as they are created if set to #t. The argument on–off deactivates the automatic display in all views of geometry entities as they are created if set to #f. When deactivated, new entities are only created in the active view. An example of where this is useful is when rendered and wireframe views of the same part are used at the same time. The default for on–off is #t. This only affects entities created after this procedure is called. To change the display state of previously created entities, use the extensions entity:erase and entity:display for those entities.

on–off activates/deactivates the automatic display in all views of geometry entities.

Limitations: None

Example:

```
; env:set–auto–display
; Disable the automatic display of
; newly created entities.
(env:set–auto–display #f)
;; ()
; Enable the automatic display of
; newly created entities.
(env:set–auto–display #t)
;; ()
```

env:set–default–color

Scheme Extension: Colors

Action: Sets the default color for newly–created entities.

Filename: scm/scmext/env_scm.cxx

APIs: None

Syntax: (**env:set–default–color** color=0 1 0)

Arg Types: color color

Returns: unspecified

Errors: None

Description: Sets the display color for newly created entities. This is an RGB color. By default, this value is #[color 0 1 0]. In order for the results to be visible, the entity should be reset with entity:set–color.

color is any of the available RGB colors.The valid colors settings are:

Black = 0 . . . . . = #[color 0 0 0]
Red = 1 . . . . . = #[color 1 0 0]
Green = 2 . . . . . = #[color 0 1 0]
Blue =3 . . . . . = #[color 0 0 1]
Cyan = 4 . . . . . = #[color 0 1 1]
Yellow = 5 . . . . . = #[color 1 1 0]
Magenta = 6 . . . . . = #[color 1 0 1]
White = 7 . . . . . = #[color 1 1 1]

Limitations: None

Example:

```
; env:set–default–color
; Create a block.
(define block1
    (solid:block (position 12 0 5)
    (position 24 10 –8)))
;; block1
; Get the default color as an rgb value.
(env:default–color)
;; #[color 0 1 0]
; Set the default color for newly created entities
; to cyan using an integer.
(env:set–default–color 4)
;; ()
; Get the default color as an rgb value.
(env:default–color)
;; #[color 0 1 1]
(define second
    (solid:sphere (position 4 4 4) 8))
;; second
; Set the default color as an rgb value.
(env:set–default–color (color:rgb 0.8 0.5 0.2))
;; ()
; Get the default color as an rgb value.
(env:default–color)
;; #[color 0.8 0.5 0.2]
; View the first entity in new default color.
(entity:set–color block1 (env:default–color))
;; ()
```

env:set–highlight–color

Scheme Extension: Colors, Highlighting

Action: Sets the highlight color.

Filename: scm/scmext/env_scm.cxx

APIs: None

Syntax: (**env:set–highlight–color** color=1 1 1)

Arg Types:	color	color

Returns: unspecified

Errors: None

Description: Sets the highlight color. This is an RGB color. By default, this value is #[color 1 1 1]. The highlight color setting affects all views. A change in highlight color affects entities that have highlighting enabled. Use the Scheme extension entity:set–highlight to enable highlighting for an entity.

color is any of the available RGB colors.The valid colors settings are:

Black = 0 . . . . . = #[color 0 0 0]
Red = 1 . . . . . = #[color 1 0 0]
Green = 2 . . . . . = #[color 0 1 0]
Blue =3 . . . . . = #[color 0 0 1]
Cyan = 4 . . . . . = #[color 0 1 1]
Yellow = 5 . . . . . = #[color 1 1 0]
Magenta = 6 . . . . . = #[color 1 0 1]
White = 7 . . . . . = #[color 1 1 1]

Limitations: None

Example:

```
; env:set–highlight–color
; Create a block.
(define block1
    (solid:block (position 12 0 5)
    (position 24 10 –8)))
;; block1
(env:highlight–color)
;; #[color 1 1 1]
; Set the display color for highlighted
; entities as an integer.
(env:set–highlight–color 4)
;; ()
; Get the default color as an rgb value.
(env:highlight–color)
;; #[color 0 1 1]
(define faces1
    (entity:faces block1))
;; faces1
(define entity (entity:set–highlight
    (car faces1) #t))
;; entity
```

env:set–point–size

Scheme Extension: Viewing, Scheme Interface

Action: Sets the displayed size of a point.

Filename: scm/scmext/env_scm.cxx

APIs: None

Syntax: `(env:set–point–size size=10)`

Arg Types: size integer

Returns: unspecified

Errors: None

Description: The argument size specifies how large or small a point is displayed on the screen; the default value is 10. The argument size is not related to the text font point size.

Limitations: None

Example:

```
; env:set–point–size
; Create a point.
(define point1 (point (position 5 4 10)))
;; point1
(env:point–size)
;; 10
; Set the display size for a point.
(env:set–point–size 20)
;; ()
; Get the display size for a point.
(env:point–size)
;; 20
```

env:set–point–style

Scheme Extension: Viewing, Scheme Interface

Action: Sets the display style for a point.

Filename: scm/scmext/env_scm.cxx

APIs: None

Syntax: `(env:set–point–style style–string=x)`

Arg Types: style–string string

Returns: unspecified

Errors: None

Description: The argument style–string specifies the style of the point; the default value is x. Valid options include:

Valid options include:

“x” or “X”, creates a point in the shape of an x.

“+” or “.”, creates a point in the shape of a + (plus) or a . (period) respectively.

“o” or “O”, creates a point in the shape of a square.

Limitations: None

Example:

```
; env:set–point–style
; Set the display style of a point to an x.
(env:set–point–style ”x”)
;; ()
; Define a point.
(define point1 (point (position –5 –5 –5)))
;; point1
; Set the display style of a point to a +.
(env:set–point–style ”+”)
;; ()
; Define a point.
(define point2 (point (position 0 0 0)))
;; point2
; Set the display style of a point to a square.
(env:set–point–style ”o”)
;; ()
; Define a point.
(define point3 (point (position 5 5 5)))
;; point3
```

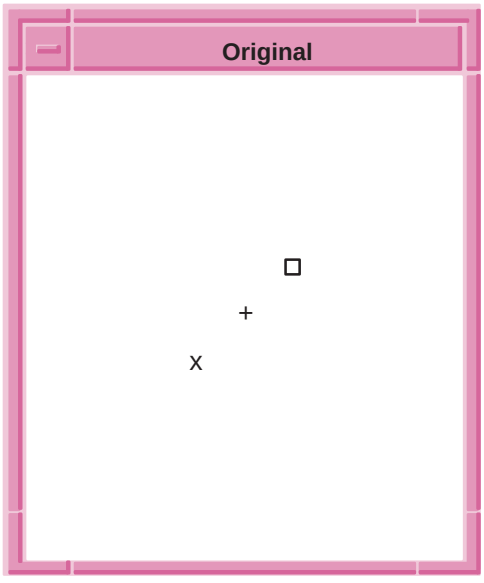

Figure 4-3. env:set–point–style

event

Scheme Extension: Picking

Action: Creates a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: `(event x y button view [state])`

Arg Types:

x	integer
y	integer
button	integer
view	view
state	string

Returns: pick–event

Errors: None

Description: Pick–events are created from user mouse/keyboard input using (read–event) or are generated from data using (event).

The argument x specifies the *x*–screen position in pixels.

The argument y specifies the *y*–screen position in pixels.

The argument button (integers 1, 2, or 3) represents the button used on a mouse click.

The argument view specifies any valid view or window handle.

The argument state specifies a list of strings for the event. Valid string options include “left”, “right”, “middle”, “shift”, “control”, or “alt”.

Limitations: None

Example:

```
; event
; Create a new view.
(define view (view:dl))
;; view
; Create pick event 1 from data.
(event 100 100 1 view)
;; #[pick–event 100 100 1 3080666 0]
; Create pick event 2 from data.
(event 100 100 1 view (list ”left” ”shift”))
;; #[pick–event 100 100 1 3080666 257]
```

event:alt?

Scheme Extension: Picking

Action: Determines if the Alt key is pressed in a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event:alt?** event)

Arg Types: event pick–event

Returns: boolean

Errors: None

Description: This extension returns #t if the event specifies that the Alt key is pressed; otherwise, it returns #f.

event specifies the event used and it is read using (read–event) or is generated from data using (event).

Limitations: None

Example:
```
; event:alt?
; Create a new view.
(define view (view:dl))
;; view
; Create an event from data.
(define event (event 100 100 1 view
    (list ”left” ”alt”)))
;; event
; Determine if the event specified that the
; ”alt” key is pressed.
(event:alt? event)
;; #t
; Determine if the event specified that the
; ”alt” key is pressed.
(event:alt? (read–event))
;; #f
; Returns #t if alt key pressed with selection
; key on mouse
; Returns #f if any or no other key pressed
; with selection key on mouse (except alt)
```

event:button

Scheme Extension: Picking

Action: Gets the button from a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event:button** event)

Arg Types:	event	pick–event

Returns: integer

Errors: None

Description: This extension returns the number of the mouse button pressed as an integer. The mouse buttons are represented as button 1 (left), button 2 (middle), or button 3 (right).

event specifies the type of event used, and it is read using (read–event) or is generated from data using (event)

Limitations: None

Example:
```
; event:button
; Create a new view.
(define view (view:dl))
;; view
; Create an event from data.
(define event (event 100 100 1 view
    (list ”left” ”alt”)))
;; event
; Determine which mouse buttons were pressed.
(event:button event)
;; 1
; Determine which mouse buttons were pressed.
(event:button (read–event))
;; 1
; Returns 1 if left mouse button
; Returns 2 if middle mouse button
; Returns 3 if right mouse button
```

event:control?

Scheme Extension: Picking

Action: Determines if the Ctrl key is pressed in a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event:control?** event)

Arg Types: event pick–event

Returns: boolean

Errors: None

Description: This extension returns #t if the event specifies that the control key is pressed; otherwise, it returns #f.

event specifies the type of event used, and it is read using (read–event) or is generated from data using (event)

Limitations: None

Example:

```
; event:control?
; Create a new view.
(define view (view:dl))
;; view
; Create an event from data.
(define event (event 100 100 1 view
    (list ”left” ”control”)))
;; event
; Determine if the event specified that the
; ”control” key is pressed.
(event:control? event)
;; #t
; Determine if the event specified that the
; ”control” key is pressed.
(event:control? (read–event))
;; #f
; Returns #t if “control” key pressed with selection
; key on mouse
; Returns #f if any or no other key pressed
; with selection key on mouse (except the “control”
; key)
```

event:left?

Scheme Extension: Picking

Action: Determines if the left mouse button is pressed in a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event:left?** event)

Arg Types: event pick–event

Returns: boolean

Errors: None

Description: This extension returns #t if the event specifies that the left mouse button is pressed; otherwise, it returns #f.

event specifies the event used, and it is using (read–event) or is generated from data using (event).

Limitations: None

Example:

```
; event:left?
; Create a new view.
(define view (view:dl))
;; view
; Create an event from data.
(define event (event 100 100 1 view
    (list ”left” ”control”)))
;; event
; Determine if the left mouse button
; was pressed during the event.
(event:left? event)
;; #t
; Determine if the event specified that
; the left key is pressed.
(event:left? (read–event))
;; #t
; Returns #t if left mouse key pressed.
; Returns #f if any other mouse key pressed.
```

event:middle?

Scheme Extension: Picking

Action: Determines if the middle mouse button is pressed in a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event:middle?** event)

Arg Types: event — pick–event

Returns: boolean

Errors: None

Description: This extension returns #t if the event specifies that the middle mouse button is pressed; otherwise, it returns #f.

event specifies the event used, and it is read using (read–event) or is generated from data using (event).

Limitations: None

Example:

```
; event:middle?
; Create a new view.
(define view (view:dl))
;; view
; Create an event from data.
(define event (event 100 100 1 view
    (list ”middle” ”control”)))
;; event
; Determine if the middle mouse button
; was pressed during the event.
(event:middle? event)
;; #t
; Determine if the event specified that
; the middle key is pressed.
(event:middle? (read–event))
;; #t
; Returns #t if middle mouse key pressed
; Returns #f if any other mouse key pressed
```

event:ray

Scheme Extension: Picking

Action: Gets the pick ray from a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event:ray** event)

Arg Types: event pick–event

Returns: ray

Errors: None

Description: This extension returns the pick ray specified by an event. event specifies the event used.

Limitations: None

Example:

```
; event:ray
; Create a view to use as input to the event.
(define view1 (view:dl))
;; view1
; Create an imaginary event from data instead of
; doing a (define event1 (read–event)).
(define event1 (event 100 100 1 view1)
    (list ”right” ”shift”))
;; event1
; Determine a ray from the event
(define ray1 (event:ray event1))
;; ray1
; ray1 =>
; #[ray (–33.25 33.25 500) (0 0 –1)]
```

event:right?

Scheme Extension: Picking

Action: Determines if the right mouse button is pressed in a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event:right?** event)

Arg Types: event pick–event

Returns: boolean

Errors: None

Description: This extension returns #t if the event specifies that the right mouse button was pressed, #f otherwise. Event specifies the event used. This can be read using (read–event) or generated from data using (event).

event specifies the event used, and it is read using (read–event) or is generated from data using (event).

Limitations: None

Example:

```
; event:right?
; Create a new view.
(define view (view:dl))
;; view
; Create an event from data.
(define event (event 100 100 1 view
    (list ”right” ”control”)))
;; event
; Determine if the right mouse button
; was pressed during the event.
(event:right? event)
;; #t
; Determine if the event specified that
; the right key is pressed.
(event:right? (read–event))
;; #t
; Returns #t if right mouse key pressed
; Returns #f if any other mouse key pressed
```

event:shift?

Scheme Extension: Picking

Action: Determines if the Shift key is pressed in a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event:shift?** event)

Arg Types: event pick–event

Returns: boolean

Errors: None

Description: This extension returns #t if the event specifies that the shift key is pressed; otherwise, it returns #f.

event specifies the event used, and it is read using (read–event) or is generated from data using (event).

Limitations: None

Example:

```
; event:shift?
; Create a new view.
(define view (view:dl))
;; view
; Create an event from data.
(define event (event 100 100 1 view
    (list ”right” ”shift”)))
;; event
; Determine if the event specified that
; the shift key was pressed.
(event:shift? event)
;; #t
; Determine if the event specified that
; the shift key is pressed.
(event:shift? (read–event))
;; #t
; Returns #t if shift key was pressed with
; selection key on mouse.
; Returns #f if any or no other key pressed with
; selection key on mouse (except shift).
```

event:view

Scheme Extension: Picking, Viewing

Action: Gets the view from a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event:view** event)

Arg Types:	event	pick–event

Returns: view

Errors: None

Description: This extension returns the view specified by an event. event specifies the event used, and it is read using (read–event) or is generated from data using (event).

event specifies the event used, and it is read using (read–event) or is generated from data using (event).

Limitations: None

Example:

```
; event:view
; Define a new view.
(define view (view:dl))
;; view
; Create an event from data.
(define event (event 100 100 1 view
    (list ”right” ”shift”)))
;; event
; Determine in which view the event occurred.
(event:view event)
;; #[view 1075519376]
; Determine if the event specified that
; the selection key is pressed.
(event:view (read–event))
;; #[view 1075519376]
; If selection key is pressed in the view.
```

event:x

Scheme Extension: Picking

Action: Gets the *x*–coordinate from a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event:x** event)

Arg Types: event pick–event

Returns: integer

Errors: None

Description: This extension returns the *x*–coordinate of the pick–event in pixels. event specifies the event used, and it is read using (read–event) or is generated from data using (event).

event specifies the event used, and it is read using (read–event) or is generated from data using (event).

Limitations: None

Example:

```
; event:x
; Create an event from data.
(define view1 (view:dl))
;; view1
; Define an event in the new view.
(define event1 (event 100 150 1 view1
    (list ”right” ”shift”)))
;; event1
; Determine the X location of the event.
(event:x event1)
;; 100
; Determine if the event specified an X location.
(event:x (read–event))
;; 334
; Location of event.
; Define an event in the new view.
(define event2 (event 100 150 1 view1
    (list ”right” ”shift”)))
;; event2
; Determine the X location of the event.
(event:x event2)
;; 100
; Determine if the event specified an X location.
(event:x (read–event))
;; 513
; Location of event.
```

event:y

Scheme Extension: Picking

Action: Gets the *y*–coordinate from a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event:y** event)

Arg Types: event — pick–event

Returns: integer

Errors: None

Description: This extension returns the *y*–coordinate of the pick–event in pixels. event specifies the event used, and it is read using (read–event) or is generated from data using (event).

Limitations: None

Example:
```
; event:y
; Create an event from data.
(define view1 (view:dl))
;; view1
; Define an event in the new view.
(define event1 (event 100 150 1 view1
    (list ”right” ”shift”)))
;; event1
; Determine the Y location of the event.
(event:y event1)
;; 150
; Determine if the event specified an Y location.
(event:y (read–event))
;;111
; Location from pick
```

event?

Scheme Extension: Picking

Action: Determines if a Scheme object is a pick–event.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**event?** object)

Arg Types:	object	scheme–object

Returns: boolean

Errors: None

Description: This extension returns #t if the object is a pick–event; otherwise, it returns #f.

object is an input Scheme object.

Limitations: None

Example:

```
; event?
; Create an event from data.
(define view1 (view:dl))
;; view1
(define event1 (event 100 100 1 view1
    (list ”left” ”control”)))
;; event1
; Determine if the object is an event.
(event? event1)
;; #t
```

filter:color

Scheme Extension: Filtering, Colors

Action: Creates an entity–filter that tests on the basis of color.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: (**filter:color** color)

Arg Types: color color

Returns: entity–filter

Errors: None

Description: The filter color created defines the color of entity used for any particular filter operation. color specifies a color integer (0 – 7) or an rgb color value (double, double, double).

Additional filter types can be created. Refer to filter:type and filter:display.

Limitations: None

Example:

```
; filter:color
; Create solid block 1.
(define block1
    (solid:block (position 10 0 10)
    (position 20 30 40)))
;; block1
; block1 => #[entity 2 1]
; Create linear edge 2.
(define edge1 (edge:linear (position 0 0 0)
    (position 10 10 10)))
;; edge1
; edge1 => #[entity 3 1]
; Create circular edge 3.
(define edge2 (edge:circular (position 0 0 0) 20))
;; edge2
; edge2 => #[entity 4 1]
; Change the color of the entities so far to red.
(entity:set-color (part:entities) 1)
;; ()
; Create solid sphere 4.
(define sphere1 (solid:sphere
    (position 20 30 40) 30))
;; sphere1
; sphere1 => #[entity 5 1]
; Create solid sphere 5.
(define cyl1 (solid:cylinder
    (position 40 0 0) (position 5 5 5) 8))
;; cyl1
; cyl1 => #[entity 6 1]
; Create linear edge 6.
(define edge3 (edge:linear (position 0 50 0)
    (position 50 50 0)))
;; edge3
; edge3 => #[entity 7 1]
; Create spline edge 7.
(define edge4 (edge:spline (list
    (position 20 20 20) (position 10 20 30)
    (position 50 40 10))))
;; edge4
; edge4 => #[entity 8 1]
(zoom-all)
;; #[view 1075640874]
; OUTPUT Original
```

```
; Apply a green filter and obtain the entities.
(filter:apply (filter:color 2) (part:entities))
;; (#[entity 5 1] #[entity 6 1]
;; #[entity 7 1] #[entity 8 1])
```

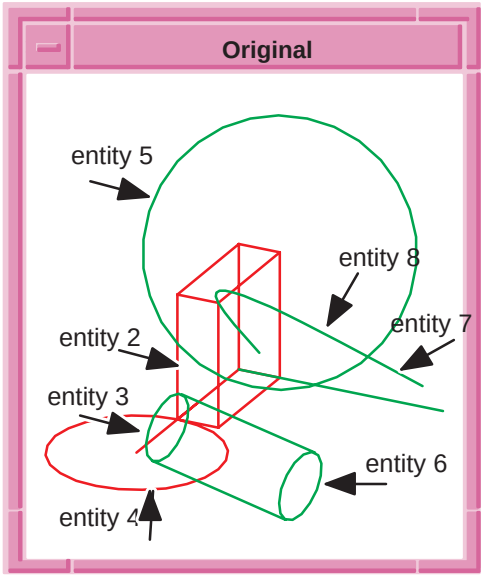

Figure 4-4. Filter:color

filter:display

Scheme Extension: Filtering, Viewing

Action: Creates a filter entity that tests on the basis of a display status.

Filename: scm/scmext/dent_scm.cxx

APIs: None

Syntax: (**filter:display**)

Arg Types: None

Returns: entity–filter

Errors: None

Description: Refer to Action.

Limitations: None

Example:

```
; filter:display
; Create solid block 1.
(define block1
    (solid:block (position 10 0 10)
    (position 20 30 40)))
;; block1
; block1 => #[entity 2 1]
; Create linear edge 2.
(define edge1 (edge:linear (position 0 0 0)
    (position 10 10 10)))
;; edge1
; edge1 => #[entity 3 1]
; Create circular edge 3.
(define edge2 (edge:circular (position 0 0 0) 20))
;; edge2
; edge2 => #[entity 4 1]
; Change the color of the entities so far to red.
(entity:set–color (part:entities) 1)
;; ()
; Create solid sphere 4.
(define sphere1 (solid:sphere
    (position 20 30 40) 30))
;; sphere1
; sphere1 => #[entity 5 1]
; Create solid sphere 5.
(define cyl1 (solid:cylinder
    (position 40 0 0) (position 5 5 5) 8))
;; cyl1
; cyl1 => #[entity 6 1]
; Create linear edge 6.
(define edge3 (edge:linear (position 0 50 0)
    (position 50 50 0)))
;; edge3
; edge3 => #[entity 7 1]
; Create spline edge 7.
(define edge4 (edge:spline (list
    (position 20 20 20) (position 10 20 30)
    (position 50 40 10))))
;; edge4
; edge4 => #[entity 8 1]
; Apply a green filter and obtain the entities.
(filter:apply (filter:color 2) (part:entities))
;; (#[entity 5 1] #[entity 6 1]
```

```
;; #[entity 7 1] #[entity 8 1])
; Apply a display filter to list the entities
; that are displayed.
(filter:apply (filter:display) (part:entities))
;; (#[entity 2 1] #[entity 3 1] #[entity 4 1]
;; #[entity 5 1] #[entity 6 1] #[entity 7 1]
;; #[entity 8 1])
```

gvector:view

Scheme Extension: Viewing

Action: Creates a gvector in the view coordinate system.

Filename: scm/scmext/vwgm_scm.cxx

APIs: None

Syntax: (**gvector:view** x y z [view=active])

Arg Types:

x	real
y	real
z	real
view	view

Returns: gvector

Errors: None

Description: x specifies the value of the x-coordinate. y specifies the value of the y-coordinate. z specifies the value of the z –coordinate. The optional view specifies which view to set the direction; the default is the active system coordinates. This extension returns the view in active system coordinates.

Limitations: None

Example:

```
; gvector:view
; Create a gvector in the active view's
; coordinate system.
(gvector:view 4 4 5)
;; #[gvector 4.88865347297876 –0.833035035958355
;; 5.69272516902042]
```

history

Scheme Extension: History and Roll

Action: Gets a history stream from an entity, part, ID, or string.

Filename: scm/scmext/roll_scm.cxx

APIs: api_get_history_from_entity, api_part_get_distribution_mode

Syntax:
```
(history part | entity | history-id
    [part-id] | "default")
```

Arg Types:

part	part
entity	entity
history–id	integer
part–id	integer
“default”	string

Returns: history | boolean

Errors: None

Description: Three types of history exist: part histories, entity histories, and the default history.

Part histories contain rollback information for all entities in a part, unless those entities have entity history. Obtain this history with:
`(history part)`

Entity histories can be attached to top–level entities via an attribute and contain history for all the entity and all the subordinate entities. Obtain entity history with:

`(history entity)`

history–id **gives the id of the history.**

part–id **gives the id of the part history.**

Finally, the default history contains history for entities without history and in a part without part history. Obtain the default history with:

`(history “default”)`

If an entity with entity history is deleted or rolled before it’s creation, its history still exists. The part contains an array of histories, which can be obtained with:

`(history history-id part-id)`

Unless specified otherwise with option:set, each part generally has its own history stream associated with it.

Limitations: None

Example:

```
; history
; Define a new part.
(define part1 (part:new))
;; part1
; Define a new view for the part.
(define view1 (view:dl part1))
;; view1
(env:set-active-part part1)
;; ()
; Define a 2nd new part.
(define part2 (part:new))
;; part2
; Define a 2nd new view for the part.
(define view2 (view:dl part2))
;; view2
; Get the default history.
(history "default")
;; #[history 0 0]
(history part1)
;; #f
(history part2)
;; #f
```

history:get–modified–faces

Scheme Extension: History and Roll

Action: Finds faces that have been deleted, created, or modified since a particular state.

Filename: scm/scmext/roll_scm.cxx

APIs: api_find_named_state, api_get_modified_faces

Syntax:

```
(history:get-modified-faces mod-type-string
    [delta-state-name-string=null] [history])
```

Arg Types:		
	mod–type–string	string
	delta–state–name–string	string
	history	history

Returns: (entity ...) | (integer ...)

Errors: rollback state not on history stream main branch

Description: This extension is intended to find lists of faces that have been created, deleted, or modified between the named delta state and the current state of the given history stream. If no history stream is specified, the default stream is used. If no delta state is specified, the beginning of the history stream is used.

The mod–type string argument controls which search is done. Allowed values are “created”, “deleted”, or “modified”. Only the first character is examined. If “created” or “modified” is specified, a list of the actual faces is returned; if “deleted” is specified, however, this extension returns a list of the ID tags associated with each of the faces, with the value –1 indicating that no ID tag has been associated with the corresponding face.

For the purposes of this extension, a face is not considered modified if its associated attributes or bounding box changes. A face is considered modified if one of its “contained” entities is modified. These contained entities are its surface, loops, coedges, edges (and associated curves) and vertices (and associationed apoints).

One intended use of the underlying api is to allow customers to determine which faces in a model need to be re–rendered.

delta–state–name–string is the name of the delta state

The optional history specifies the history stream to save; if this is not specified the default history stream is used. This is used to get the delta state.

Limitations: None

Example:

```
; history:get-modified-faces
; Create solid block 1,
(roll:name-state "block")
;; "block"
(define block1 (solid:block
    (position -20 -20 -20) (position 30 30 30)))
;; block1
(history:get-modified-faces "c" "block")
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;;  #[entity 6 1] #[entity 7 1] #[entity 8 1])
(history:get-modified-faces "m" "block")
;; ()
(history:get-modified-faces "d" "block")
;; ()
(roll:name-state "round")
;; "round"
(blend:round (list-ref
    (entity:edges block1) 9) 10 "fix")
;; #t
(history:get-modified-faces "c"  "round")
;; (#[entity 21 1] #[entity 22 1] #[entity 23 1])
(history:get-modified-faces "m" "round")
;; (#[entity 4 1] #[entity 6 1] #[entity 8 1])
(history:get-modified-faces "d" "round")
;; (-1 -1)
```

history:load

Scheme Extension: History and Roll

Action: Loads one or more history streams and associated entities from a file.

Filename: scm/scmext/part_scm.cxx

APIs: api_get_active_entities, api_pm_create_part, api_restore_history

Syntax: (**history:load** filename [textmode=#t] [part=active])

Arg Types:

filename	string
textmode	boolean
part	part

Returns: (part ...)

Errors: None

Description: This extension loads one or more history streams from a file. Each history stream (and its associated entities) is placed in a distinct part. When placing the first stream, the extension examines the part specified as an argument. If the part is empty, then its history stream is replaced by the first stream and the first stream's entities are placed in it. If the part is not empty, then a new part is created into which the first stream and its entities are placed. New parts are created for all additional streams. A list of the parts into which streams were placed is returned by the extension.

filename specifies a filename in the current working directory or the entire path that includes the filename.

textmode describes the type of file to load. If textmode is specified as #t, the data in filename is saved in text mode as a .sat file. If textmode is #f, then filename is loaded as a binary file (sab). If textmode is not specified, then the mode is determined by the extension of the filename. If the filename string ends in .sab or .SAB, then the file is saved in binary mode; otherwise, the file is saved in text mode.

part is an input entity.

If part–load–path is set to a list of strings, they are interpreted as directories to be searched for the part file. part–load–path is to part–load as load–path is to load. Set the part–load–path in an initialization file.

Limitations: There is a known limitation when using the Scheme interface on the Macintosh platform. The Scheme extension part:load accepts pathnames. However, it will not accept an absolute path on the Macintosh unless the disk name is only one letter long. Relative pathnames work as expected. For example, if your Macintosh disk drive name is *reptile*, the absolute pathname :

reptile:spatial:3dt:examples:block.scm

is converted to:

e:spatial/3dt/examples/block.scm

before it is processed. Because there is no disk drive named *e*, the part:load extension produces a ”file not found” error.

Example:

```
; history:load
; turn on distributed history
(part:set-distribution-mode #t)
;; #t
; create a block
(define block1 (solid:block 0 0 0 10 10 10))
;; block1
; create both blocks
(define block2 (solid:block 20 20 20 30 25 22))
;; block2
; save blocks with history
(history:save "myblock.sat" (history
(env:active-part)))
;; #t
; delete all entities in the part
(part:clear)
;; #t
; read in blocks with history
(history:load "myblock.sat")
;; (#[part 1])
; should be 2 blocks
(part:entities)
;; (#[entity 3 1] #[entity 4 1])
; roll to state with only first block
(roll)
;;  -1
; should be 1 block
(part:entities)
;; (entity 4)
```

history:save

Scheme Extension: History and Roll

Action: Saves all entities, entity–id, and roll information associated with a history stream to a file.

Filename: scm/scmext/part_scm.cxx

APIs: api_save_history, api_save_version

Syntax:

```
(history:save filename [text-mode=#t]
    [save-type="entire"] [history])
```

Arg Types:	filename	string
	text–mode	boolean
	save–type	string
	history	history

Returns: boolean

Errors: None

Description: The filename specifies a filename to be saved in the current working directory or specifies the path that includes the filename to be saved to a directory.

If textmode is #t, the data is saved in text mode. If textmode is not specified, then the mode is determined by the extension of the filename. If the filename string ends in .sab or .SAB, then the file is saved in binary mode; otherwise, the file is saved in text mode (.sat).

The save–type string can be one of ”entire–history”, ”mainline”, or ”active–only”. Only the first character of the string is examined. The non–default ”mainline” and ”active–only” options limit the amount of roll information stored in the history data: ”mainline” indicates that rolled branches are not saved while ”active–only” saves only active entities (and their creation history data) and their id information.|

Files can be saved in the format of an earlier version by setting the save_version option.

The optional history specifies the history stream to save; if this is not specified the default history stream is used.

Limitations: None

Example:

```
; history:save
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Save the history_stream of the currently-active
; part to the named file.
(history:save "myblock.sat"
    (history (env:active-part)))
;; #t
; Set the save_version option to ACIS 1.7.
(option:set "save_version" 107)
;; 800
; Save the solid block again in ACIS 1.7 format.
(history:save "myblock17.sat"
    (history (env:active-part)))
;; #t
```

history:size

Scheme Extension: History and Roll

Action: Returns the size of the given history stream.

Filename: scm/scmext/roll_scm.cxx

APIs: api_get_history_size

Syntax: (**history:size** hist)

Arg Types:	hist	history

Returns: real

Errors: None

Description: Refer to action.

Limitations: None

Example:

```
; history:size
; turn on distributed history
(part:set-distribution-mode #t)
;; #t
; give a name to history stream of active part
(define h (history (env:active-part)))
;; h
; and find its size
(history:size h)
;; 64
; create a block
(define block1 (solid:block 0 0 0 10 10 10))
;; block1
(history:size h)
;; 192
; create second block
(define block2 (solid:block 20 20 20 30 25 22))
;; block2
(history:size h)
;; 256
```

journal:abort

Scheme Extension: Scheme AIDE Application

Action: Terminates the journal single stepping process.

Filename: scm/scmext/jrl_scm.cxx

APIs: None

Syntax: (**journal:abort**)

Arg Types: None

Returns: unspecified

Errors: None

Description: When stepping, journal:abort terminates the current load without executing the rest of the commands in the file.

Limitations: None

Example:

```
; journal:abort
; Turn journaling on.
(journal:on)
;; ”j001.jrl”
; Create solid block 1.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Create solid block 2.
(define block2
    (solid:block (position 5 10 15)
    (position 10 20 35)))
;; block2
; Turn journaling off
(journal:off)
;; ()
; Save the resulting journal.
(journal:save ”j008.jrl”)
;; ””j008.jrl””
; Clear the part.
(part:clear)
;; #t
; Turn on single step mode.
(journal:step #t)
;; (journal–step . #t)
; Load the journal file to recreate
; and redisplay the solid block.
(journal:load ”j001.jrl”)
;; ()
(journal:abort)
;; (journal–abort . #t)
```

journal:append

Scheme Extension: Scheme AIDE Application

Action: Opens an existing journal file and appends additional journal data to the end of the file.

Filename: scm/scmext/jrl_scm.cxx

APIs: None

Syntax: (**journal:append** [filename])

Arg Types:	filename	string
Returns:	unspecified	
Errors:	None	
Description:	This extension opens the optional filename, if it exists, where all future commands are journaled to the existing file. The time and date of the append to the existing file are indicated in the journal. If the filename is not specified, a unique name is created after reading the current directory. The unique name is numerically sequenced from the last journal file created or named j(last number+1).jrl.	
Limitations:	None	

Example:

```
; journal:append
; Start journalling and create some geometry.
(journal:on 'test.jrl)
;; "test.jrl"
; Create solid block 1.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Create solid block 2.
(define block2 (solid:block
    (position 5 10 15) (position 10 20 35)))
;; block2
; Turn journaling off
(journal:off)
;; ()
; Save the resulting journal.
(journal:save 'test.jrl)
;; "test.jrl"
; Clear the part.
(part:clear)
;; #t
; Load the journal file to recreate
; and redisplay the solid block.
(journal:load 'test.jrl)
;; "test.jrl"
; Initiate the append command.
(journal:append 'test.jrl)
;; "test.jrl"
; Add additional changes to blocks.
(define color1 (entity:set–color block1 1))
;; color1
(define color2 (entity:set–color block2 4))
;; color2
; Turn journaling off and save new version.
(journal:off)
;; ()
(journal:save 'test.jrl)
;; "test.jrl"
; Load journal file to verify successful appending.
(journal:load 'test.jrl)
; the contents of the journal are displayed
;; test.jrl
```

journal:load

Scheme Extension: Scheme AIDE Application

Action: Loads a journal file one line at a time, journaling each command as it is executed.

Filename: scm/scmext/jrl_scm.cxx

APIs: None

Syntax: `(journal:load filename)`

Arg Types: filename string

Returns: unspecified

Errors: None

Description: The journal:load extension loads an existing journal file and runs each of the commands contained in that file. Each line is journaled if journaling is on.

This extension works like the load primitive, except the file is evaluated one line at a time instead of all at once with the Scheme load primitive. Encountered errors do not abort the load operation and are reported in the command window. This extension is useful for debugging Scheme files as well as for rerunning the commands given in another Scheme session.

Note *An error in the loaded file does not abort the evaluation.*

The mouse–up–hook and mouse–down–hook are no longer journaled, because the results are incomplete without journaling all of the mouse moves as well.

The journal:load extension permits single stepping through a loaded file one line at a time. (journal:step #t) turns it on and should be run before loading the journal file. In addition, (option:set "timing" #t) can be used to show the execution time for each command in the output window.

Before loading, the directory where the load file is found is added to the part–load–path

Limitations: None

Example:

```
; journal:load
; Turn journaling on.
(journal:on)
;; ”j002.jrl”
; Create solid block 1.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Create solid block 2.
(define block2 (solid:block
    (position 5 10 15) (position 10 20 35)))
;; block2
; Turn journaling off
(journal:off)
;; ()
; Save the resulting journal.
(journal:save ”j002.jrl”)
;; ”j002.jrl”
; Clear the part.
(part:clear)
;; #t
; Load the journal file to recreate
; and redisplay the solid block.
(journal:load ”j002.jrl”)
;; ”j002.jrl”
; (journal:on)
; ”j010.jrl”
; (define block1 (solid:block
; (position 0 0 0) (position 10 10 10)))
; block1
; (define block2 (solid:block
; (position 5 10 15) (position 10 20 35)))
; block2
; (journal:off)
```

journal:off

Scheme Extension: Scheme AIDE Application

Action: Closes the current journal file and turns off journaling.

Filename: scm/scmext/jrl_scm.cxx

APIs: None

Syntax: `(journal:off)`

Arg Types: None

Returns: unspecified

Errors: None

Description: This extension turns off journaling. All extensions executed after journaling has been turned off are not retained in the journaling file.

Limitations: None

Example:

```
; journal:off
; Turn journaling on.
(journal:on)
;; "j004.jrl"
; Create solid block 1.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Create solid block 2.
(define block2
    (solid:block (position 5 10 15)
    (position 10 20 35)))
;; block2
; Save the resulting journal.
; Turn journaling off
(journal:off)
;; ()
```

journal:on

Scheme Extension: Scheme AIDE Application

Action: Closes the current journal file and opens a new journal file.

Filename: scm/scmext/jrl_scm.cxx

APIs: None

Syntax: `(journal:on [filename])`

Arg Types: filename string

Returns: unspecified

Errors: None

Description: This extension opens the optional specified filename, if it exists, and all future commands are journaled to the existing file. If filename is not specified, the extension creates a unique name after reading the current directory. The unique name is numerically sequenced from the last journal file created or named j(last number+1).jrl. If the file exists, it is truncated to zero length.

Limitations: None

Example:

```
; journal:on
; Close the current journal file and open
; a new journal file called new_jrl.
(journal:on ”new_jrl”)
;; ”new_jrl”
; Create solid block 1.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Create solid block 2.
(define block2
    (solid:block (position 5 10 15)
    (position 10 20 35)))
;; block2
; Turn journaling off
(journal:off)
;; ()
; Save the resulting journal.
(journal:save ”new_jrl”)
;; ”new_jrl”
; Clear the part.
(part:clear)
;; #t
; Load the journal file to recreate
; and redisplay the solid block.
(journal:load ”new_jrl”)
;; “new_jrl”
; (journal:on ”new_jrl”)
; ”new_jrl”
; (define block1 (solid:block
; (position 0 0 0) (position 10 10 10)))
; block1
; (define block2 (solid:block
; (position 5 10 15) (position 10 20 35)))
; block2
; (journal:off)
```

journal:pause

Scheme Extension: Scheme AIDE Application

Action: Disables journaling temporarily but leaves the journal file open.

Filename: scm/scmext/jrl_scm.cxx

APIs: None

Syntax: (**journal:pause**)

Arg Types: None

Returns: unspecified

Errors: None

Description: This extension does not record in the journal any procedures evaluated while the journal file is paused.

Limitations: None

Example:

```
; journal:pause
; Turn journaling on.
(journal:on)
;; “j012.jrl”
; Create solid block 1.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Temporarily disable journaling.
(journal:pause)
;; ()
; Create solid block 2.
(define block2
    (solid:block (position 5 10 15)
    (position 10 20 35)))
;; block2
; Resume journaling.
(journal:resume)
;; ()
; Create solid block 3.
(define block3
    (solid:block (position –5 10 –15)
    (position 10 –20 35)))
;; block3
; Turn journaling off
(journal:off)
;; ()
; Save the resulting journal.
(journal:save ”j012.jrl”)
;; ”j012.jrl”
```

```
; Clear the part.
(part:clear)
;; #t
; Load the journal file to recreate
; and redisplay the solid block.
(journal:load "j012.jrl")
;; "j012.jrl"
; (journal:on)
; (define block1 (solid:block
; (position 0 0 0) (position 10 10 10)))
; block1
; (journal:pause)
; (journal:resume)
; ()
; (define block3 (solid:block
; (position –5 10 –15) (position 10 –20 35)))
; block3
; (journal:off)
; "j012.jrl"
```

journal:resume

Scheme Extension: Scheme AIDE Application

Action: Resumes journaling in the journal file again after pausing.

Filename: scm/scmext/jrl_scm.cxx

APIs: None

Syntax: (**journal:resume**)

Arg Types: None

Returns: unspecified

Errors: None

Description: Refer to Action.

Limitations: None

Example:

```
; journal:resume
; Turn journaling on.
(journal:on)
;; "j015.jrl"
```

```
; Create solid block 1.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Temporarily disable journaling.
(journal:pause)
;; ()
; Create solid block 2.
(define block2
    (solid:block (position 5 10 15)
    (position 10 20 35)))
;; block2
; Resume journaling.
(journal:resume)
;; ()
; Create solid block 3.
(define block3
    (solid:block (position –5 10 –15)
    (position 10 –20 35)))
;; block3
; Turn journaling off
(journal:off)
;; ()
; Save the resulting journal.
(journal:save ”j015.jrl”)
;; ”j015.jrl”
; Clear the part.
(part:clear)
;; #t
; Load the journal file to recreate
; and redisplay the solid block.
(journal:load ”j015.jrl”)
;;
;; (journal:on)
;; ”j015.jrl”
;; (define block1 (solid:block
;; (position 0 0 0) (position 10 10 10)))
;; block1
;; (journal:pause)
;; (journal:resume)
;; ()
:: (define block3 (solid:block
;; (position –5 10 –15) (position 10 –20 35)))
;; block3
```

```
;; (journal:off)
;; "j015.jrl"
```

journal:save

Scheme Extension: Scheme AIDE Application

Action: Saves the current journal to a file.

Filename: scm/scmext/jrl_scm.cxx

APIs: None

Syntax: (**journal:save** filename)

Arg Types:	filename	string

Returns: unspecified

Errors: None

Description: filename specifies the name of the file in which to save the journal. This extension is ignored if journaling is not active. Refer to journal:on for more details. An error is generated if the filename is the same as the current journal file.

Limitations: None

Example:

```
; journal:save
; Turn journaling on.
(journal:on)
;; ”j004.jrl”
; Create solid block 1.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Turn journaling off
(journal:off)
;; ()
; Save the resulting journal.
(journal:save ”j004.jrl”)
;; ”j004.jrl”
; Clear the part.
(part:clear)
;; #t
; Load the journal file to recreate
; and redisplay the solid block.
(journal:load ”j004.jrl”)
;; ”j004.jrl”
; contents of journal file are displayed
; (journal:on)
; ”j004.jrl”
; (define block1 (solid:block
; (position 0 0 0) (position 10 10 10)))
; block1
; (journal:off)
; j004.jrl
```

journal:step

Scheme Extension: Scheme AIDE Application

Action: Enables or disables single stepping of journal file on and off.

Filename: scm/scmext/jrl_scm.cxx

APIs: None

Syntax: (**journal:step** value)

Arg Types: value boolean

Returns: boolean

Errors: None

Description: journal:step sets a flag to control stepping through the journal file. When stepping is on, the system waits for input after printing, but before executing each line. A single return causes the line to be executed. Anything else is evaluated and the system waits for more input. This allows one to setup demos and to debug scheme scripts one line at a time.

To run a demo, enter (journal:step #f), but include (journal:step #t) at points in the script where you want to stop and talk. Extra commands can be executed to show things that are not a part of the canned demo. Or press the return key a few times to step through it slowly. Re–enter free running mode with (journal:step #f).

Limitations: None

Example:

```
; journal:step
; Turn journaling on.
(journal:on)
;; ”j015.jrl”
; Create solid block 1.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Create solid block 2.
(define block2
    (solid:block (position 5 10 15)
    (position 10 20 35)))
;; block2
; Switch the step mode to off.
(journal:step #f)
;; (journal–step . #f)
; Create solid block 3.
(define block3
    (solid:block (position –5 10 –15)
    (position 10 –20 35)))
;; block3
; Turn journaling off
(journal:off)
;; ()
; Save the resulting journal.
(journal:save ”j015.jrl”)
;; ”j015.jrl”
; Clear the part.
(part:clear)
;; #t
; Turn on step mode.
(journal:step #t)
;; (journal–step . #t)
; Load the journal file to recreate
; and redisplay the solid block.
(journal:load ”j015.jrl”)
;; #f
```

load–dll

Scheme Extension: Scheme AIDE Application

Action: Loads a dynamically–linked library for the NT platform.

Filename: scm/scmext/scm_dll.cxx

APIs: None

Syntax: `(load-dll filename)`

Arg Types: filename string

Returns: unspecified

Errors: None

Description: This extension loads a dynamically–linked library at run time and attaches any Scheme primitives (extensions) defined in the library to the parser. This enables users to define their own Scheme extensions in DLLs, and then selectively load those DLLs at run time as needed. This extension returns an empty list.

filename specifies the name of the file.

Limitations: NT platforms only.

Example:
```
; load-dll
; Load a dynamically linked library.
(load-dll "mylib.dll")
```

part

Scheme Extension: Part Management, Physical Properties

Action: Gets a Scheme part from its identification number.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: `(part id)`

Arg Types: id integer

Returns: part | boolean

Errors: None

Description: id specifies a positive integer assigned to a part entity. This extension returns the part; otherwise, it returns #f if the part is not found. Use the env:set–active–part extension to establish the active part.

Limitations: None

Example:
```
; part
; define a part
(define part1 (part:new))
;; part1
; Find a part from its ID number.
(part 3)
;; #f
; Find a part from its ID number.
(part 2)
;; #[part 2]
```

part:clear

Scheme Extension: Part Management

Action: Deletes all entities from the active part.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: (**part:clear** [part=active])

Arg Types:	part	part

Returns: boolean

Errors: None

Description: This extension does not reset the entity counter to 1. Use the env:set–active–part extension to establish the active part.

Note *Roll back is not supported after processing this extension. The entity counter is not reset to 1.*

Limitations: None

Example:
```
; part:clear
; Create a block.
(define block1
    (solid:block (position 0 0 0)
    (position 5 10 15)))
;; block1
; Create a sphere
(define sphere1 (solid:sphere (position 0 0 0) 10))
;; sphere1
; Delete all entities in the active part.
(part:clear)
;; #t
```

part:clear–selection

Scheme Extension: Entity

Action: Clear the list of currently selected entities.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: `(part:clear–selection [part=active])`

Arg Types: part part

Returns: unspecified

Errors: None

Description: This extension return a list of all currently selected entities in a part. If the optional part is not specified, then the active part is assumed. Use the env: set-active-part extension to establish the active part.

Note Use part:clear-selection to deselect all selected entities.

Limitations: None

Example:

```
; part:clear–selection
; Create a new part
(define part1 (part:new))
;; part1
; Create a new block
(define block (solid:block 0 0 0 1 1 1))
;; block
; Create a new sphere
(define sphere (solid:sphere 0 0 0 1))
;; sphere
; Select block and sphere
(entity:select block sphere)
;; ()
; Clear selection
(part:clear–selection)
;; ()
```

part:close

Scheme Extension: Part Management

Action: Closes (deletes) a part.

Filename: scm/scmext/part_scm.cxx

APIs: api_pm_delete_all_states, api_pm_delete_part

Syntax: `(part:close [part=active])`

Arg Types: part part

Returns: unspecified

Errors: None

Description: This extension deletes a part and all of the entities in it. If the optional part is not specified, then the active part is closed. Use the env:set–active–part extension to establish the active part. The part is removed from env:parts. The display list associated with this part is deleted along with all rendering contexts contained in this part. It closes all windows and files associated with views that depict the entities in the part's display list. These views are removed from the env:views list.

Note *Roll back is not supported for this part after processing this extension.*

Limitations: None

Example:

```
; part:close
; Create a new part
(define part1 (part:new))
;; part1
(part:close part1)
;; ()
```

part:debug

Scheme Extension: Debugging, Part Management

Action: Gets information about the entities in a part and writes it to the active output port.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: `(part:debug [part=active])`

Arg Types: part part

Returns: unspecified

Errors: None

Description: This extension prints debug information to the output port. The first column specifies the entity number. The second column indicates the entity type. The remaining columns specify the address location of the entity when it is displayed.

Limitations: None

Example:

```
; part:debug
; create some entities to delete
(define block1
    (solid:block (position 0 0 0)
    (position 5 10 15)))
;; block1
; Print the debug information.
(part:debug)
; --- Entity ID table ---
; 1, rh_background, 4024f010, top level
; 2, body, 40255c00, top level
; --- End of ID table ---
;; ()
(define edges1 (entity:edges block1))
;; edges1
; Print the debug information.
(part:debug)
; --- Entity ID table ---
; 1, body, 401f6038, top level
; 2, edge, 401f29e0, not top level
; 3, edge, 401f28d8, not top level
; 4, edge, 401f295c, not top level
; 5, edge, 401f29e0, not top level
; 6, edge, 401f2a64, not top level
; 7, edge, 401f2930, not top level
; 8, edge, 401f2abc, not top level
; 9, edge, 401f2a38, not top level
; 10, edge, 401f29b4, not top level
; 11, edge, 401f2a90, not top level
; 12, edge, 401f2904, not top level
; 13, edge, 401f2a0c, not top level
; 14, edge, 401f2a0c, not top level
; --- End of ID table ---
;; ()
```

part:delete–rendering–context

Scheme Extension: Part Management, Rendering Control

Action: Deletes a rendering context from a part.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: `(part:delete–rendering–context [part=active] name)`

Arg Types:		
	part	part
	name	string

Returns: boolean

Errors: None

Description: This removes the rendering context, name, from the specified part. If no part is supplied, the rendering context is deleted from the active part. Any views that are associated with the rendering context are also deleted while their associated windows and files are closed. If name specifies an invalid rendering context, this extension has no effect.

Limitations: None

Example:

```
; part:delete–rendering–context
; Create a new part.
(define part1 (part:new))
;; part1
; Create a new view.
(define view1 (view:dl))
;; view1
; Set the context of the view.
(view:set–context view1 ”dl”)
;; #[view 1075606912]
; Get the context of the view.
(view:context view1)
;; ”dl_context”
; Delete the rendering context for the part
(part:delete–rendering–context part1 ”dl”)
;; #t
; Get the context of the view.
(view:context view1)
;; ”dl_context”
```

part:entities

Scheme Extension: Part Management, Debugging

Action: Gets a list of all top–level entities in a part.

Filename: scm/scmext/part_scm.cxx

APIs: api_pm_part_entities

Syntax: (**part:entities** [part=active] [filter=NULL])

Arg Types:

part	part
filter	entity–filter

Returns: (entity ...)

Errors: None

Description: This extension obtains a list of the top–level entities from the specified part. If the optional part is not specified, it obtains a list of the top–level entities from the active part. Use the optional filter to control the kinds of entities that are returned by this extension.

An ENTITY is top level when making a call to api_get_owner returns itself. Also, every ENTITY contains an owner method. This method would return the next higher ENTITY. If that object is the top level ENTITY, then this pointer is returned. This means that if a FACE does not point to an owning SHELL, this FACE is top level for that model. A BODY is normally top level, but in some cases, there are others that are the top level ENTITY.

filter is an entity–filter that is a procedural object that selects entities from an entity–list.

Limitations: None

Example:

```
; part:entities
; List the entities in the active part.
(part:entities)
;; (#[entity 1 1])
(define block1
    (solid:block (position 0 0 0)
    (position 5 10 15)))
;; block1
(part:entities)
;; (#[entity 1 1] #[entity 2 1])
(define edges1 (entity:edges block1))
;; edges1
; Create a point.
(define point1 (point (position 0 0 0)))
;; point1
; Get a list of the top level entities in a part.
(part:entities)
;; (#[entity 1 1] #[entity 2 1] #[entity 15 1])
; Get a list of the solid entities in a part.
(part:entities (filter:type "solid?"))
;; (#[entity 2 1])
```

part:get–distribution–mode

Scheme Extension: Part Management, Debugging

Action: Returns whether distributed history is on or off.

Filename: scm/scmext/roll_scm.cxx

APIs: api_part_get_distribution_mode

Syntax: (**part:get–distribution–mode**)

Arg Types: None

Returns: boolean

Errors: None

Description: Returns TRUE or FALSE depending on whether or not distributed history is on or off.

Limitations: None

Example:

```
; part:get–distribution–mode
; get the distribution mode of the part
(part:get–distribution–mode)
;; #f
```

part:load

Scheme Extension: Part Management

Action: Loads a part from a file into the active part.

Filename: scm/scmext/part_scm.cxx

APIs: api_pm_load_part

Syntax:
```
(part:load filename [textmode=#t] [part=active]
    [with–history=#f])
```

Arg Types:

filename	string
textmode	boolean
part	part
with–history	boolean

Returns: (entity ...)

Errors: None

Description: This extension is a merge function. This means that the restore does not replace entities but adds them to the present working session. The list of entities is returned.

filename specifies a filename in the current working directory or specifies the path that includes the filename. If the optional part is not specified, then the entities in the part file are merged into the active part; otherwise, they are merged into the specified part.

textmode describes the type of file to load. If textmode is specified as #t, the data in filename is saved in text mode as an .sat file. If textmode is #f, then filename is loaded as a binary file. If textmode is not specified, then the mode is determined by the extension of the filename. If the filename string ends in .sab or .SAB, then the file is saved in binary mode; otherwise, the file is saved in text mode.

After specifying the textmode, a second Boolean argument indicates whether to restore rollback history data. History data may only be restored into an empty part. The default, #f, does not restore history.

If part–load–path is set to a list of strings, they are interpreted as directories to be searched for the part file. part–load–path is to part–load as load–path is to load. Set the part–load–path in an initialization file.

When a part is saved using part:save and a filename, the filename becomes the new name for the part.

Limitations: There is a known limitation when using the Scheme interface on the Macintosh platform. The Scheme extension part:load accepts pathnames. However, it will not accept an absolute path on the Macintosh unless the disk name is only one letter long. Relative pathnames work as expected.

For example, if your Macintosh disk drive name is reptile, the absolute pathname:

reptile:spatial:3dt:examples:block.scm

is converted to:

e:spatial/3dt/examples/block.scm

before it is processed. Because there is no disk drive named e, the part:load extension produces a “file not found” error.

Example:

```
; part:load
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Save the part under the specified name.
(part:save ”eraseme.sat”)
;; #t
; Delete all entities from the part.
(part:clear)
;; #t
; Create another solid block.
(define block2
    (solid:block (position –20 –30 –10)
    (position –5 –10 –15)))
;; block2
; Load the saved part (block).
(part:load ”eraseme.sat”)
;; (#[entity 4 1] #[entity 5 1])
```

part:modified?

Scheme Extension: Part Management

Action: Determines if a part has been modified.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: `(part:modified? [part=active])`

Arg Types: part — part

Returns: boolean

Errors: None

Description: Refer to Action.

Limitations: None

Example:

```
; part:modified
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Define a new part.
(part:new)
;; #[part 2]
; Determine if a part has been modified.
(part:modified? (part 2))
;; #f
```

part:name

Scheme Extension: Part Management

Action: Gets the name of a part.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: `(part:name [part=active])`

Arg Types: part — part

Returns: string | boolean

Errors: None

Description: The name of the part is used as the default for the part, if the part (filename) is not specified. If the part is not found, this extension returns #f. When a part is saved using part:save and a filename, the filename becomes the new name for the part.

Limitations: None

Example:
```
; part:name
; Set the name of the currently-active part.
(part:set-name "block.sat")
;; ()
; Get the name of the currently-active part.
(part:name)
;; "block.sat"
; Save the part under the specified name.
(part:save "block2.sat")
;; #t
; Get the name of the currently-active part.
(part:name)
;; "block2.sat"
```

part:new

Scheme Extension: Part Management

Action: Creates a new part.

Filename: scm/scmext/part_scm.cxx

APIs: api_pm_create_part

Syntax: (**part:new** [size=1009])

Arg Types:	size	integer

Returns: part

Errors: None

Description: The optional size, which must be a prime number, specifies how large to make the table that stores the entities in the part. This number does not limit the maximum number of entities that can be stored, but if a part has many more than this number of defined entities, performance slows when the application looks up the entities from Scheme. If the number is much larger than the number of entities referenced from Scheme, then more memory is used than is actually needed.

The default size is DEFAULT_PART_SIZE, which is defined in the file pmhusk/part_utl.hxx (value is 1009).

This procedure is affected by two of the option:set arguments.

option:set entity_history when #t, causes history to be kept in a separate stream for each top–level entity in the part. The history streams are attached via an ATTRIB_HISTORY. ATTRIB_HISTORY also takes care of merging history streams when bodies are merged via Booleans and other operations. The default for entity_history is #f.

option:set entity_history when #f causes a single history stream to be used for the whole part, and history data for entities in the part goes to the default stream. This stream is distinct from the histories of all other parts.

The with–history option allows one to save entity–specific history. (That is, history attached to the entity via attribute.)

Limitations: None

Example:

```
; part:new
; Define a new part
(define part1 (part:new))
;; part1
; Set the part as the active part in the environment.
(env:set-active-part part1)
;; ()
```

part:save

Scheme Extension: Part Management, SAT Save and Restore

Action: Saves all entities in a part to a file.

Filename: scm/scmext/part_scm.cxx

APIs: api_pm_save_part, api_save_version

Syntax:

```
(part:save [filename=partname.sat] [textmode=#t]
    [part=active] [with-history=#f]
    [mainline-only=#f])
```

Arg Types:		
	filename	string
	textmode	boolean
	part	part
	with–history	boolean
	mainline–only	boolean

Returns: boolean

Errors: None

Description: The optional filename specifies a filename to be saved in the current working directory or specifies the path that includes the filename to be saved to a directory. If no filename is specified, this extension uses the name given to the part with the part:set–name extension. The optional part specifies the part to save; the default is the active part.

If textmode is #t, the data is saved in text mode If textmode is not specified, then the mode is determined by the extension of the filename. If the filename string ends in .sab or .SAB, then the file is saved in binary mode; otherwise, the file is saved in text mode.

After setting textmode, a second Boolean specifies whether to save rollback history data. The default, #f, does not save history. A third boolean specifies whether rolled branches should be saved. The default, #f, for this argument indicates the entire history should be saved.

The saved file can be restored at any time. When a part is saved using part:save and a filename, the filename becomes the new name for the part.

Part files can be saved in the format of an earlier version by setting the save_version option.

The with–history option allows one to save entity–specific history. (That is, history attached to the entity via attribute.) It specifies whether or not to save roll–back history data.

mainline–only specifies whether rolled back branches should be saved.

Limitations: None

Example:

```
; part:save
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Save the currently–active part to the named file.
(part:save ”myblock.sat”)
;; #t
; Set the save_version option to ACIS 1.7.
(option:set ”save_version” 107)
;; 700
; Save the solid block again in ACIS 1.7 format.
(part:save ”myblock17.sat”)
;; #t
```

part:save–selection

Scheme Extension: Part Management, SAT Save and Restore

Action: Saves a list of entities to a file.

Filename: scm/scmext/part_scm.cxx

APIs: api_save_entity_list_file, api_save_version

Syntax:

```
(part:save–selection ent–list filename [textmode=#f]
    [with–history=#f])
```

Arg Types:

ent–list	entity | entity ...
filename	string
textmode	boolean
with–history	boolean

Returns: boolean

Errors: None

Description: ent–list is a list of entities.

The filename specifies a filename to be saved in the current working directory or specifies the path that includes the filename to be saved to a directory. The ent–list is the list of entities to be saved in the output file.

If textmode is #t, the data is saved in text mode If textmode is not specified, then the mode is determined by the extension of the filename. If the filename string ends in .sab or .SAB, then the file is saved in binary mode; otherwise, the file is saved in text mode.

The saved file can be restored at any time. Part files can be saved in the format of an earlier version by setting the save_version option.

The with–history option allows one to save entity–specific history. (That is, history attached to the entity via attribute.)

Limitations: None

Example:

```
; part:save-selection
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Save the currently-active part to the named file.
(part:save-selection block1 "block.sat" #t)
;; #t
; Clear the part
(part:clear)
;; #t
; Restore the saved entity
(define restore (part:load "block.sat"))
;; restore
```

part:selection

Scheme Extension: Part Management

Action: Returns a list of currently selected entities.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: `(part:selection [part=active])`

Arg Types: part part

Returns: (entity ...)

Errors: None

Description: This extension return a list of all currently selected entities in a part. If the optional part is not specified, then the active part is assumed. Use the env: set-active-part extension to establish the active part.

Note Use part:clear-selection to deselect all selected entities.

Limitations: None

Example:

```
; part:selection
; Create a new part
(define part1 (part:new))
;; part1
; Create a new block
(define block (solid:block 0 0 0 1 1 1))
;; block
; Create a new sphere
(define sphere (solid:sphere 0 0 0 1))
;; sphere
; Select block and sphere
(entity:select block sphere)
;; ()
; Get current selection
(part:selection)
;; block sphere
```

part:set–distribution–mode

Scheme Extension: Part Management, Debugging

Action: Turns distributed history on or off.

Filename: scm/scmext/roll_scm.cxx

APIs: api_part_set_distribution_mode

Syntax: (**part:set–distribution–mode** on–off)

Arg Types: on–off boolean

Returns: boolean

Errors: None

Description: Turns distributed history on or off. After distribution is set (either on or off) it cannot be changed. Distributed history is off by default, and the first bulletin created for a part's history stream locks it in this state. This Scheme extension replaces distributed_history.

Limitations: None

Example:

```
; part:set–distribution–mode
; set the distribution mode of the part
(part:set–distribution–mode #t)
;; #t
```

part:set–name

Scheme Extension: Part Management

Action: Sets the name of a part.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: (**part:set–name** [name=id name] [part=active])

Arg Types:

name	string
part	part

Returns: unspecified

Errors: None

Description: The optional name specifies the desired string identification of the part. If the part is saved without a filename, the extension uses the identification name assigned to the part as the saved part filename.

Limitations: None

Example:

```
; part:set–name
; Define a new part
(define part1 (part:new))
;; part1
; Set the part as the active part in the environment.
(env:set–active–part part1)
;; ()
; Set the name of the currently–active part.
(part:set–name ”block.sat”)
;; ()
; Save again in ACIS1.7 format
(option:set ”save_version” 107)
;; 700
(part:save ”block17.sat”)
;; #t
```

part:views

Scheme Extension: Part Management, Viewing

Action: Gets all of the views displaying a part.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: `(part:views [part=active])`

Arg Types: part — part

Returns: (view ...)

Errors: None

Description: Refer to Action.

Limitations: None

Example:

```
; part:views
; Define a new part
(define part1 (part:new))
;; part1
; Set the part as the active part in the environment.
(env:set–active–part part1)
;; ()
; define some views
(define view1 (view:dl))
;; view1
(define view2 (view:dl))
;; view2
; Get all views displaying a part.
(part:views)
;; (#[view 10755338721] #[view 10755339071])
```

part?

Scheme Extension: Part Management

Action: Determines if a Scheme object is a part.

Filename: scm/scmext/part_scm.cxx

APIs: None

Syntax: `(part? object)`

Arg Types: object — scheme–object

Returns: boolean

Errors: None

Description: Refer to Action.

object is an input scheme object.

Limitations: None

Example:
```
; part?
; Define a new part
(define part1 (part:new))
;; part1
; Set the part as the active part in the environment.
(env:set–active–part part1)
;; ()
; Determine if part1 is a part.
(part? part1)
;; #t
```

pick:aperture

Scheme Extension: Picking

Action: Gets the dimensions of a pick aperture.

Filename: scm/scmext/pick_scm.cxx

APIs: None

Syntax: (**pick:aperture**)

Arg Types: None

Returns: pair | boolean

Errors: None

Description: This extension returns the pick aperture's dimensions in the form (aperture–x. aperture–y). The default is a width of 15 pixels and a height of 15 pixels. The aperture specifies the size of the window surrounding the cursor; items inside the aperture window could be selected by a read–event. Use the extension pick:set–aperture to change the aperture size.

Limitations: None

Example:
```
; pick:aperture
; Get the dimensions of the pick aperture.
(pick:aperture)
;; (15 . 15)
(pick:set–aperture 25 25)
;; ()
(pick:aperture)
;; (25 . 25)
```

pick:edge

Scheme Extension: Picking, Model Topology

Action: Gets an edge from a pick event.

Filename: scm/scmext/pick_scm.cxx

APIs: None

Syntax: (**pick:edge** {event | pick–ray} [filter])

Arg Types:

event	pick–event
pick–ray	ray
filter	entity–filter

Returns: entity | boolean

Errors: None

Description: If the picked entity has subcomponents (e.g., the edges of a solid), this extension returns the subcomponent as the entity. This extension returns #f if no valid object is registered within the pick:aperture by the pick action.

event specifies the pick event, typically, a mouse click.

pick–ray specifies the ray.

The optional filter specifies the entity to include with the pick.

Limitations: None

Example:

```
; pick:edge
; Create a solid block.
(define block1
    (solid:block (position –10 –10 0)
    (position 25 25 25)))
;; block1
; Extract an edge from a pick–event.
; Select an edge of the block with the mouse.
(define edge1 (pick:edge (read–event)))
;; edge1
; Returns #f if no edge is picked.
; Select an edge of the block with the mouse.
(define edge2
    (pick:edge (read–event)
    (filter:type ”edge:linear?”)))
;; edge2
; Returns #f if no edge is picked.
```

pick:entity

Scheme Extension: Picking, Entity

Action: Gets an entity from the pick event.

Filename: scm/scmext/pick_scm.cxx

APIs: None

Syntax: `(pick:entity event [filter] [depth])`

Arg Types:

event	pick–event
filter	entity–filter
depth	integer

Returns: entity | boolean

Errors: None

Description: When picking several connected entities (e.g., all edges of a face), pick one entity, the face, and find the others by following model topology pointers.

event specifies the event used, typically, a mouse click.

The optional filter specifies the entity to include with the pick event. depth

The optional depth specifies an integer that allows the picking of subcomponents of an object. If depth is 0, top level entities are picked. If depth is greater than 0, subcomponents (for example, edges of a solid) are picked.

This extension returns #f if no valid object is registered by the pick action.

Limitations: None

Example:

```
; pick:entity
; Create a solid block.
(define block1
    (solid:block (position –10 –10 0)
    (position 25 25 25)))
;; block1
; Get an entity from pick–event 1.
(define pick1 (pick:entity (read–event)))
; Using the mouse, select the block.
;; pick1
; Returns #f if nothing selected with the mouse.
(define list (entity:edges pick1))
;; list
; Get an entity from pick–event 2.
(define pick2 (pick:entity (read–event) 2))
; Using the mouse, select the block.
;; pick2
; Returns #f if nothing selected with the mouse.
```

pick:face

Scheme Extension: Picking, Model Topology

Action: Gets a face from a pick event.

Filename: scm/scmext/pick_scm.cxx

APIs: None

Syntax: (**pick:face** {event | pick–ray} [face–index] [filter])

Arg Types:

event	pick–event
pick–ray	ray
face–index	integer
filter	entity–filter

Returns: face | (face ...) | boolean

Errors: None

Description: event specifies the event used, typically, a mouse click.

pick–ray specifies the ray.The pick event's ray starts at the eye position in the viewing plane and goes through the pick event point. Therefore, faces closer to the eye position are listed first. The ray extends infinitely in the positive direction. Only faces that are in the forward direction from the ray are picked.

The optional filter specifies the entity to include with the pick event.

The optional face–index specifies the *n*th face that the ray pierces. If face–index is not specified, the extension returns a list of all faces in the order that the ray pierces them. If face–index is greater than the number of faces hit by the ray, the last face to be hit is returned.

This extension returns #f if no face was registered by the pick event.

Limitations: None

Example:

```
; pick:face
; Extract a face from a pick–event.
; Create a solid block.
(define block1
    (solid:block (position –10 –10 0)
    (position 25 25 25)))
;; block1
; Get a list of the faces so we know what the
; entities could be
(define list (entity:faces block1))
;; list
; Extract planar–face 1 from a pick–event.
(define face1 (pick:face (read–event)))
; Using the mouse, select a face on the block.
;; face1
; Returns #f in nothing selection.
; Extract planar–face 2 from a pick–event using
; a face–index integer.
(define face2 (pick:face (read–event) 2))
; Using a mouse, select a face on the block.
;; face2
; Returns #f in nothing selection.
```

pick:in–region

Scheme Extension: Picking

Action: Gets all entities in a region specified by two pick events.

Filename: scm/scmext/pick_scm.cxx

APIs: None

Syntax: (**pick:in–region** event1 event2 [filter])

Arg Types:

event1	pick–event
event2	pick–event
filter	entity–filter

Returns: (entity ...)

Errors: None

Description: event1 and event2 specify the event used, typically, a mouse click, and they must be in the same view. filter

The optional filter specifies the entity to include. The empty list is returned if no entities are picked. The entity is included in the region if any portion of it is within the region.

Limitations: None

Example:

```
; pick:in–region
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position –5 –5 –5)
    (position –30 0 –30) 10))
;; cyl1
; Create a block.
(define block1
    (solid:block (position 5 5 5)
    (position 20 0 20)))
;; block1
; Get all the entities lying within a rectangular
; region defined by two pick–events.
; Create a rectangle that contains block
; within it.
; Define the first corner of a rectangle.
(define event1 (read–event))
;; event1
; Define the first corner to be the start
; position.
```

```
(define start (pick:position event1))
;; start
; Use rubberbanding to visually aid corner selection.
; Create a rectangle rubberband driver.
(rbd:rectangle #t start)
;; #[rbd–driver 401b1df0]
; Define the second corner of rectangle.
(define event2 (read–event))
;; event2
; Define the second corner to be the end
; position.
(define end (pick:position event2))
;; end
; Remove the rubberbanding rectangle.
(rbd:remove–type rbd:rectangle?)
;; ()
; Using the mouse, select diagonal corners of
; the window.
(define pick (pick:in–region event1 event2))
;; pick
```

pick:position

Scheme Extension: Picking

Action: Gets the position of a pick event.

Filename: scm/scmext/pick_scm.cxx

APIs: None

Syntax:
```
(pick:position event [plane–position
    [plane–direction=xy–plane]])
```

Arg Types:

event	pick–event
plane–position	position
plane–direction	gvector

Returns: position

Errors: None

Description: This extension computes the position of the pick event mapped into the space of the active WCS, if specified; otherwise, it maps into model space coordinates. If snapping is active, the position component is snapped to the grid. Positions returned are approximate.

event specifies the event used, typically, a mouse click.

The optional plane–position specifies the position of the plane to pick.

The optional plane–direction specifies the normal direction of the plane from which to pick. The default is the *xy*–plane in model space.

Limitations: None

Example:

```
; pick:position
; Create a solid block.
(define block1
    (solid:block (position –10 –10 0)
    (position 25 25 25)))
;; block1
; Extract a position of the pick from a pick–event.
(pick:position (read–event))
; Using the mouse, select screen position 1
; corresponding approximately to the upper
; left corner of the front of the block.
;; #[position –34.3990726470947 39.1702480316162 0]
; Extract another position of the pick from a
; pick–event.
(pick:position
    (read–event) (position 0 0 0)
    (gvector 0 1 0))
; Using the mouse, select screen position 1
; corresponding approximately to the lower
; right corner of the front of the block.
;; #[position 20.2177810668945 0 –5.35901403427124]
; OUTPUT Example 1
```

```
; Second example:
; Draw a rubberbanded rectangle using
; pick:position.
; 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 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)))
        (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.
(define rectangle (rectangle start end))
;; rectangle
; #[entity 2 1] #[entity 3 1] #[entity 4 1]
; #[entity 5 1]
; Turn off rubberbanding.
(rbd:remove–type rbd:rectangle?)
;; ()
; OUTPUT Example 2
```

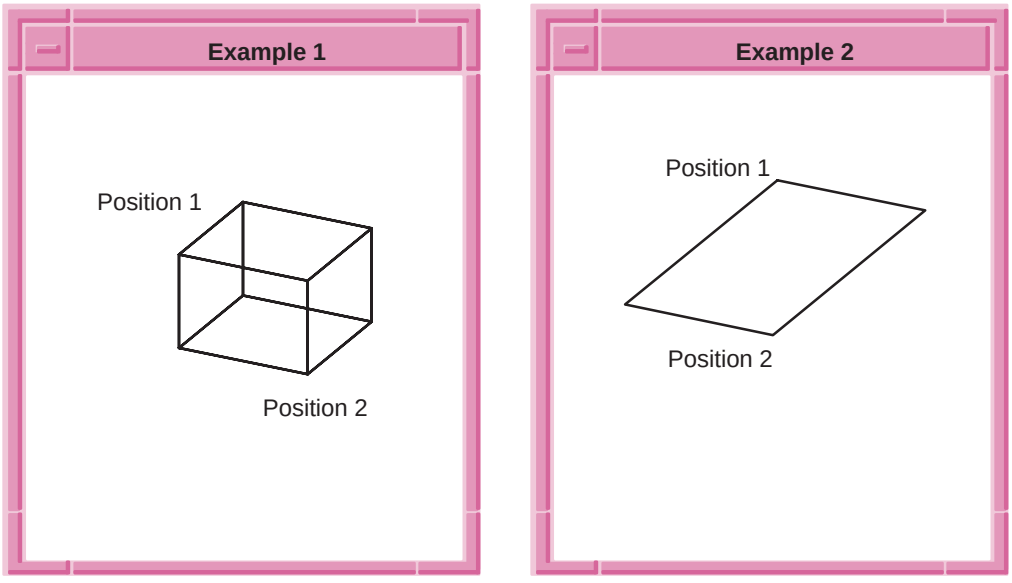

Figure 4-5. pick:position

pick:ray

Scheme Extension: Picking

Action: Gets a ray from a pick event.

Filename: scm/scmext/pick_scm.cxx

APIs: None

Syntax:
```
(pick:ray event [plane–position
    [plane–direction=xy–plane]])
```

Arg Types:

event	pick–event
plane–position	position
plane–direction	gvector

Returns: ray

Errors: None

Description: This extension computes a pick event's ray. The ray maps into the space of the active WCS, if specified; otherwise, it maps into model space coordinates. If snapping is active, the position component is snapped to the grid.

event specifies the event used, typically, a mouse click.

The optional plane–position specifies the position of the plane to pick.

The optional plane–direction specifies the normal direction of the plane from which to pick; the default is the *xy*–plane in model space.

Limitations: None

Example:

```
;; pick:ray
; Create a solid block.
(define block1
    (solid:block (position –10 –10 0)
    (position 25 25 25)))
;; block1
; Get a ray from pick–event 1.
(pick:ray
    (read–event) (position 0 0 0)
    (gvector 1 0 0))
; Using the mouse, select screen position 1
; corresponding approximately to the lower
; left corner of the front of the block.
;; #[ray (0 –29.5407 9.5048)
;; (–0.408248 0.816497 –0.408248)]
; Get a ray from pick–event 2.
(pick:ray
    (read–event) (position 0 0 0)
    (gvector 1 0 0))
; Using the mouse, select screen position 2
; corresponding approximately to the upper
; right corner of the front of the block.
;; #[ray (0 40.2082 –0.505848)
;; (–0.408248 0.816497 –0.408248)]
; OUTPUT Result
```

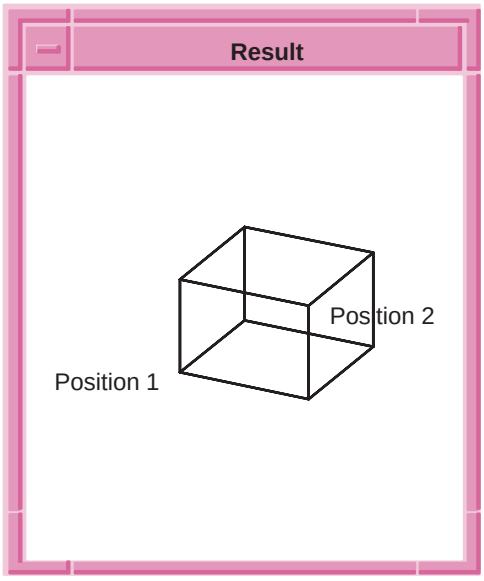

Figure 4-6. pick:ray

pick:set–aperture

Scheme Extension: Picking

Action: Sets the pick aperture in x and y.

Filename: scm/scmext/pick_scm.cxx

APIs: None

Syntax: `(pick:set–aperture dx=15 dy=15)`

Arg Types:

dx		integer
dy		integer

Returns: unspecified

Errors: None

Description: dx specifies the *x*–value in pixels.

dy specifies the *y*–value in pixels.Pick extensions return the first item found within a box of this size about the cursor position.

This extension pick:set–aperture changes the aperture size. The aperture specifies the size of the window surrounding the cursor; items inside the aperture window could be selected by a read–event. dx specifies the *x*–value in pixels. dy specifies the *y*–value in pixels. The default is a width of 15 pixels and a height of 15 pixels.

Limitations: None

Example:

```
; pick:set–aperture
; Get the dimensions of the pick aperture.
(pick:aperture)
;; (15 . 15)
(pick:set–aperture 25 25)
;; ()
(pick:aperture)
;; (25 . 25)
```

pick:vertex

Scheme Extension: Picking

Action: Gets a vertex from a pick event.

Filename: scm/scmext/pick_scm.cxx

APIs: None

Syntax: (**pick:vertex** {event | pick–ray} [filter])

Arg Types:

event	pick–event
pick–ray	ray
filter	entity–filter

Returns: vertex | boolean

Errors: None

Description: This extension gets the vertex at the end of an entity subcomponent.

event specifies the event used, typically, a mouse click.

pick–ray specifies the ray.

The optional filter specifies the entity to include with the pick event. This extension returns #f if no valid object is registered by the pick action.

Limitations: None

Example:

```
; pick:vertex
; Create a solid block.
(define block1
    (solid:block (position –10 –10 0)
    (position 25 25 25)))
;; block1
; Extract a vertex from a pick–event.
(define vertex1 (pick:vertex (read–event)))
; Select a vertex on the block (position 1)
; with the mouse.
;; vertex1
; Returns #f if nothing selected.
; OUTPUT Result
```

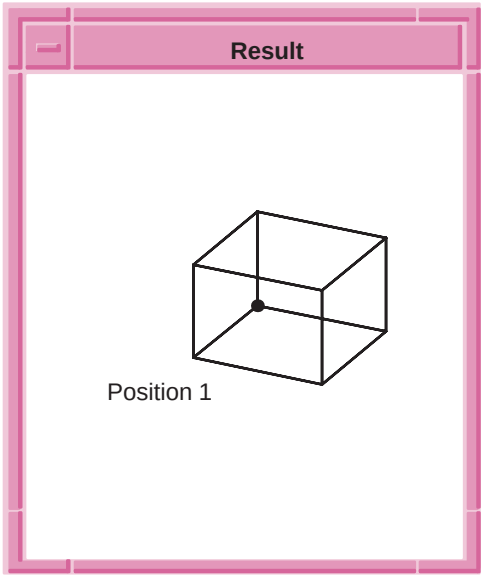

Figure 4-7. pick:vertex

position:view

Scheme Extension: Viewing

Action: Creates a new position given coordinates x, y, and z in the view coordinate system.

Filename: scm/scmext/vwgm_scm.cxx

APIs: None

Syntax: `(position:view x y z [view=active])`

Arg Types:

x	real
y	real
z	real
view	view

Returns: position

Errors: None

Description: x specifies the value of the x-coordinate.

y specifies the value of the y-coordinate.

z specifies the value of the z-coordinate.

The optional view specifies the view to create the position; the default is the active system coordinates. This extension returns the position in active system coordinates.

Limitations: None

Example:

```
; position:view
; Create a new view.
(define ISO (view:dl))
;; ISO
; Create new positions in the active view's
; coordinate system.
(position:view 0 –1 0)
;; #[position 0.182574185835055 –0.365148371670111
;; –0.912870929175277]
; Create new positions in the ISO view's
; coordinate system.
(position:view 0 –1 0 ISO)
;; #[position 0 –1 0]
```

read–event

Scheme Extension: Picking

Action: Creates a pick–event Scheme object from mouse input.

Filename: scm/scmext/evnt_scm.cxx

APIs: None

Syntax: (**read–event**)

Arg Types: None

Returns: pick–event

Errors: None

Description: This extension waits for mouse input from the application, then generates a pick–event Scheme object.

The Scheme object contains five numbers. The first and second numbers specify the *x*–position and *y*–position in pixels of the cursor at the time a mouse click event was initiated. The third number specifies the mouse button initiated; valid options are 1, 2, or 3. The fourth number specifies the view handle used for the active view where the event took place. The fifth numerically codes the state of the modifier keys.

Limitations: None

Example:

```
; read–event
; Create a solid block.
(define block1
    (solid:block (position –10 –10 –10)
    (position 10 10 10)))
;; block1
; Specify a pick–event 1.
(read–event)
; Using mouse button 1, select point on block1.
;; #[pick–event 221 367 1 1075533160 0]
; Specify pick–event 2.
(read–event)
; Using mouse button 2, select point on block1.
;; #[pick–event 326 278 2 1075533160 0]
; Specify pick–event 3.
(read–event)
; Using mouse button 3, select point on block1.
;; #[pick–event 275 213 3 1075533160 0]
```

```
; Two–step entity dragging example follows.
; Click on the block. If you click somewhere
; other than on the block, an error message
; is displayed.
(define evt (read–event))
;; evt
; Get entity portion of event.
(define ent (pick:entity evt))
;; ent
; Get original location of entity / event.
(define pos1 (pick:position evt))
;; pos1
; Test to see if object is an entity.
; Move the mouse to new position. The
; image should follow.
; When at new location, click mouse again.
(if (entity? ent)
    (begin
        (rbd:drag #t ent pos1)
        (rbd:line #t pos1)
        (define pos2 (pick:position (read–event)))
        (rbd:remove–type rbd:drag?)
        (rbd:remove–type rbd:line?)
        (entity:transform ent
            (transform:translation
            (gvector:from–to pos1 pos2) )) )
        (ui:error–dialog
            ”Object selected is not entity.”) )
;; #[entity 3 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 3 1]
; OUTPUT Result
```

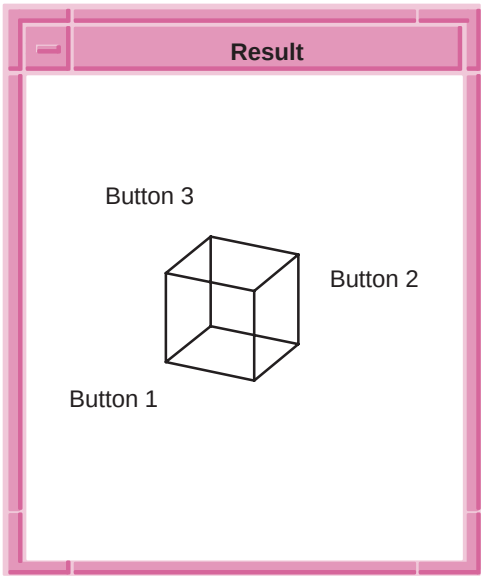

Figure 4-8. read–event

ro:color

Scheme Extension: Viewing, Colors

Action: Gets the color of a display item in the display list.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: `(ro:color item)`

Arg Types:

item	ro

Returns: color

Errors: None

Description: Returns the color of the specified display item in the display list. A display item is not part of the model and does not get saved and restored.

item is the display item.

Limitations: None

Example:

```
; ro:color
; Define two polyline list items.
(define poly1 (ro:polyline
    (list (position 0 0 0) (position 5 0 0)
    (position 0 5 0) (position 0 0 0)) #t))
;; poly1
; Color poly1 red.
(ro:set–color poly1 1)
;; ()
(define poly2 (ro:polyline
    (list (position 10 10 10)
    (position 15 10 10) (position 10 15 10)
    (position 10 10 10)) #t))
;; poly2
; Color poly2 blue.
(ro:set–color poly2 3)
;; ()
; OUTPUT Display

; Request the color of poly1 and poly2.
(ro:color poly1)
;; #[color 1 0 0]
; color 1 0 0 is red.
(ro:color poly2)
;; #[color 0 0 1]
; color 0 0 1 is blue.
```

ro:delete

Scheme Extension: Viewing

Action: Deletes the rendering object.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: (**ro:delete** rendering–object)

Arg Types: rendering–object ro

Returns: None

Errors: None

Description: Deletes the rendering object.

rendering–object is the rendering object to be deleted.

Limitations: None

Example:
```
; ro:delete
; Create a point display list item.
(define dp (ro:point (position 15 5 10)))
;; dp
(ro:delete dp)
;; ()
```

ro:display

Scheme Extension: Viewing

Action: Displays the rendering object.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: `(ro:display rendering–object)`

Arg Types:	rendering–object	ro

Returns: None

Errors: None

Description: Displays the rendering object.

rendering–object is the rendering object to be displayed.

Limitations: None

Example:
```
; ro:display
; Create a point display list item.
(define dp (ro:point (position 15 5 10)))
;; dp
(ro:erase dp)
(ro:display dp)
```

ro:erase

Scheme Extension: Viewing

Action: Erases the rendering object.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: `(ro:erase rendering–object)`

Arg Types:

rendering–object	ro

Returns: None

Errors: None

Description: Erases the rendering object.

rendering–object is the rendering object to be erased.

Limitations: None

Example:

```
; ro:display
; Create a point display list item.
(define dp (ro:point (position 15 5 10)))
;; dp
(ro:erase dp)
(ro:display dp)
```

ro:line

Scheme Extension: Viewing, Colors

Action: Creates a line.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: `(ro:line pos1 pos2)`

Arg Types:

pos1	position
pos2	position

Returns: ro

Errors: None

Description: Returns the new rendering object.

pos1 is the start position.

pos2 is the end position.

Limitations: None

Example:

```
; ro:line
; Create a line between the position
(ro:line ro (position 0 0 0) (position 3 3 3))
;; ro
```

ro:new

Scheme Extension: Viewing, Colors

Action: Creates a new rendering object.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: `(ro:new view–opt)`

Arg Types:	view–opt	string

Returns: ro

Errors: None

Description: Returns the new rendering object.

view–opt specifies the view option on the rendering object.

Limitations: None

Example:

```
; ro:new
; Create a new rendering object
(define ro (ro:new 'all'))
;; ro
```

ro:point

Scheme Extension: Viewing

Action: Creates a point item for the display list.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: `(ro:point position)`

Arg Types:	position	position

Returns: ro

Errors: None

Description: This extension creates a point display list item at the specified position. The point is displayed in all views until it is deleted. The point style is controlled by env:set-point-style and env:set-point-size.

A display item is not part of the model and does not get saved and restored. Even though a 3D position is provided for the display item, it is mapped into 2D space. A display item should then not be used for snapping or as part of a model.

Limitations: None

Example:
```
; ro:point
; Create a point display list item.
(define dp (ro:point (position 15 5 10)))
;; dp
```

ro:polyline

Scheme Extension: Viewing

Action: Creates a polyline item for the display list.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: (**ro:polyline** pos–list [fill=#f])

Arg Types:		
	pos–list	(position ...)
	fill	boolean

Returns: ro

Errors: None

Description: This extension creates a polyline display item for the display list connecting the specified positions. The polyline is displayed in all views until it is removed.

A display item is not part of the model and does not get saved and restored. Even though a 3D position is provided for the display item, it is mapped into 2D space. A display item should then not be used for snapping or as part of a model.

pos–list list of positions.

fill is an optional fill flag that indicates whether the polyline is filled.

Limitations: None

Example:

```
; ro:polyline
; Create a polyline display list item
; for a filled triangle.
(define poly (ro:polyline (list
    (position 0 0 0) (position 5 0 0)
    (position 0 5 0) (position 0 0 0)) #t))
;; poly
```

ro:text

Scheme Extension: Viewing

Action: Creates a text display list item.

Filename: scm/scmext/ro_scm.cxx

APIs: None

Syntax: (**ro:text** position text)

Arg Types:		
	position	position
	text	string

Returns: ro

Errors: None

Description: This extension creates a text display list item at the specified position. The text displays in all views until it is removed.

A display item is not part of the model and does not get saved and restored. Even though a 3D position is provided for the display item, it is mapped into 2D space. A display item should then not be used for snapping or as part of a model.

Limitations: None

Example:

```
; ro:text
; Create a text display list item.
(define dt (ro:text (position 10 0 15) ”Hello”))
;; dt
; Create a second text display list item.
(define ds (ro:text (position –10 0 15) ”Test”))
;; ds
(ro:set–color dt (color:rgb 1 0 0))
;; ()
```

roll

Scheme Extension: History and Roll

Action: Rolls to a previous or later state.

Filename: scm/scmext/roll_scm.cxx

APIs: api_pm_roll_n_states, api_pm_roll_to_state

Syntax:
```
(roll [number–of–states=–1] [name–string=NULL]
    [end] [start] [history])
```

Arg Types:

number–of–states	integer
name–string	string
history	history

Returns: integer

Errors: None

Description: The optional number–of–states specifies the number of states the model rolls. A negative number means to roll to a previous state, and a positive number means to roll to a later state.

The optional name–string identifies a user–defined name, to which the model should go. If no arguments are specified, the model rolls back one state.

There are two system–named states: start rolls to the beginning of the session, and end rolls to the end of the session. If previous states were deleted, or a limit on the maximum number of states to keep was set, start rolls to the first non–deleted state. If a state was named, (roll name–string) rolls the session back or forward to the identified name. The name is created with the roll:name–state extension.

The optional history is used to find the history stream to roll. The default is the stream associated with the active part.

This extension returns the number of steps rolled.

Limitations: None

Example:

```
; roll
; Roll back and forward
; to previously defined states.
; Create solid block 1,
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Create solid block 2.
(define block2
    (solid:block (position 15 15 15)
    (position 25 25 25)))
;; block2
; Create solid block 3.
(define block3
    (solid:block (position 30 30 30)
    (position 40 40 40)))
;; block3
; Roll back to the beginning of the session.
(roll ”start”)
;; 6
; Roll forward to the end of the session.
(roll ”end”)
;; 6
; Roll back one step.
(roll)
;; –1
; Define the name of the state.
(roll:name–state ”step”)
;; ”step”
; Roll back two states
(roll –2)
;; –2
; Roll back to the named state.
(roll ”step”)
;; 2
```

roll:back?

Scheme Extension: History and Roll

Action: Determines if there are any previous states to roll back to.

Filename: scm/scmext/roll_scm.cxx

APIs: None

Syntax: `(roll:back? [history])`

Arg Types:

history	history

Returns: boolean

Errors: None

Description: This extension returns #t if it is possible to roll back to a previous state; otherwise, it returns #f. It is not necessary to call this extension before invoking the roll procedure because that procedure automatically checks to make sure that it is possible to roll the specified number of steps. This extension helps update the user interface to reflect whether it is possible to perform a roll operation.

The optional history is used to find the history stream to check. The default is the stream associated with the active part.

Limitations: None

Example:

```
; roll:back?
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 10))
;; sphere1
; Determine if a roll back is possible.
(roll:back?)
;; #t
```

roll:debug

Scheme Extension: History and Roll, Debugging

Action: Writes information about the rollback operations to the debug file.

Filename: scm/scmext/roll_scm.cxx

APIs: None

Syntax: `(roll:debug history [level] [ent–name])`

Arg Types:

history	history
level	integer
ent–name	string

Returns: unspecified

Errors: None

Description: Writes debugging information about the history stream to the debug file.

The optional history is used to find the history stream to check. The default is the stream associated with the active part.

level indicates how much information to present, whereby the higher numbers give more information.

ent–name gives the name of an entity, as it would appear in the save file. If the specified entity or entities derived from it appear in any bulletins, additional data is written for those entities.

Limitations: None

Example:
```
; roll:debug
; Get history information for the active part.
; Note that the active part must be specified
; explicitly as the default is a background
; history stream not associated with a part.
(roll:debug (history "default") 1)
;; ()
; Returns debug information on the specified history.
```

roll:delete–all–states

Scheme Extension: History and Roll

Action: Deletes all delta states.

Filename: scm/scmext/roll_scm.cxx

APIs: None

Syntax: (**roll:delete–all–states** [history])

Arg Types:	history	history

Returns: integer

Errors: None

Description: This extension deletes all delta states to free memory used to maintain those states to ACIS (but not to the operating system). After calling this extension, it is no longer possible to roll backwards or forwards to any of the deleted states.

The optional history is used to find the history stream to prune. The default is the stream associated with the active part.

This extension returns the number of deleted states.

Limitations: None

Example:
```
; roll:delete-all-states
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 30))
;; sphere1
; Delete all the delta states.
(roll:delete-all-states)
;; 4
```

roll:delete–following–states

Scheme Extension: History and Roll

Action: Deletes all delta states after the current delta state.

Filename: scm/scmext/roll_scm.cxx

APIs: None

Syntax: (**roll:delete-following-states** [history])

Arg Types: history history

Returns: integer

Errors: None

Description: This extension deletes all delta states after the current state to free memory used to maintain the states in ACIS (but not the operating system). After calling this extension, it is no longer possible to roll forward to any of the deleted states. To avoid creation of branches, use this extension before operations that would create a new state. Clean up branches after they are created with roll:delete–active–states.

The optional part or entity is used to find the history stream to prune. The default is the stream associated with the active part.

This extension returns the number of deleted states.

Limitations: None

Example:

```
; roll:delete–following–states
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 10))
;; sphere1
; Roll back one state.
(roll –1)
;; –1
; Delete all the following delta states.
(roll:delete–following–states)
;; 0
```

roll:delete–inactive–states

Scheme Extension: History and Roll

Action: Deletes all delta states after the current one.

Filename: scm/scmext/roll_scm.cxx

APIs: None

Syntax: (**roll:delete–inactive–states** [history])

Arg Types: history history

Returns: integer

Errors: None

Description: This extension deletes all delta states not in the active path to free memory used to maintain those states in ACIS (but not the operating system). After calling this extension, it is no longer possible to roll to any of the deleted states. The active path is the set of delta states from the root to the currently active state.

The optional part or entity is used to find the history stream to prune. The default is the stream associated with the active part.

This extension returns the number of deleted states.

Limitations: None

Example:

```
; roll:delete–inactive–states
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 10))
;; sphere1
; Delete inactive delta states.
(roll:delete–inactive–states)
;; 0
```

roll:delete–previous–states

Scheme Extension: History and Roll

Action: Deletes all delta states before the current one.

Filename: scm/scmext/roll_scm.cxx

APIs: None

Syntax: `(roll:delete–previous–states [history])`

Arg Types: history history

Returns: integer

Errors: None

Description: This extension deletes all delta states before the current state to free memory used to maintain the states in ACIS (but not the operating system). After calling this extension, it is no longer possible to roll back to any of the deleted states.

The optional history is used to find the history stream to prune. The default is the stream associated with the active part.

This extension returns the number of deleted states.

Limitations: None

Example:

```
; roll:delete–previous–states
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 10))
;; sphere1
; Delete all previous delta states.
(roll:delete–previous–states)
;; 0
```

roll:forward?

Scheme Extension: History and Roll

Action: Determines if there are any states following which can be rolled forward to.

Filename: scm/scmext/roll_scm.cxx

APIs: None

Syntax: **(roll:forward?** [history])

Arg Types: history history

Returns: boolean

Errors: None

Description: This extension returns #t if it is possible to roll forward to another state; otherwise, it returns #f. It is not necessary to call this extension before invoking the roll extension because that extension automatically checks to make sure that the specified roll can be performed. This extension helps update the user interface to reflect whether it is possible to perform a roll operation.

The optional history is used to find the history stream to check. The default is the stream associated with the active part.

Limitations: None

Example:
```
; roll:forward?
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 10))
;; sphere1
(roll:forward?)
; Determine if a roll forward can be performed.
;; #f
; Roll back one state.
(roll –1)
;; –1
; Determine if a roll forward can be performed.
(roll:forward?)
;; #t
```

roll:get–logging

Scheme Extension: History and Roll

Action: Returns the state of the internal logging flag.

Filename: scm/scmext/roll_scm.cxx

APIs: None

Syntax: **(roll:get–logging)**

Arg Types: None

Returns: real

Errors: None

Description: Returns the state of the internal logging flag, which controls whether bulletins are created for rollback and error recovery.

Limitations: None

Example:

```
; roll:get–logging
; Create a solid block.
; get the current state of the internal logging flag.
(roll:get–logging)
;; #t
; shows logging is not active

; turn on internal logging
(roll:set–logging #f)
;; #t

; get the current state of the internal logging flag.
(roll:get–logging)
;; #f
; shows the internal logging is now active.

; turn on internal logging
(roll:set–logging #t)
;; #f

; get the current state of the internal logging flag.
(roll:get–logging)
;; #t
```

roll:mark–end

Scheme Extension: History and Roll

Action: Marks the end of a block of functions for rolling.

Filename: scm/scmext/roll_scm.cxx

APIs: api_pm_note_state

Syntax: (**roll:mark–end**)

Arg Types: None

Returns: integer

Errors: None

Description: Use roll:mark–start and roll:mark–end to group a series of operations as a single operation for rollback. For example, if a procedure is written to create a rectangle by creating four lines, a roll back can be inserted to undo the entire rectangle as a single state change. Insert the following before the line creation:

```
(roll:mark–start)
```

and the following after creation:

```
(roll:mark–end)
```

The end marks a block of operations that is treated as a single operation for roll back. Roll to mark–start and roll to mark–end blocks may be nested; in which case, the outermost block is treated as a single operation for roll back. This extension returns the delta state number.

Limitations: None

Example:

```
; roll:mark–end
; Mark the beginning of a block of operations.
(roll:mark–start)
;; 0
; Create some test entities.
(define sphere1 (solid:sphere (position 0 0 0) 10))
;; sphere1
(define sphere2 (solid:sphere (position 5 10 20) 10))
;; sphere2
(define sphere3 (solid:sphere (position 20 0 –5) 10))
;; sphere3
; Mark the end of a block of operations.
(roll:mark–end)
;; 0
; Roll back 1 state.
(roll)
;; –1
; All created entities should be gone.
; Roll forward 1 state.
(roll 1)
;; 1
; All created entities should be back.
```

roll:mark–start

Scheme Extension: History and Roll

Action: Marks the start of a block of functions for rolling.

Filename: scm/scmext/roll_scm.cxx

APIs: api_pm_start_state

Syntax: (**roll:mark–start**)

Arg Types: None

Returns: integer

Errors: None

Description: Use roll:mark–start and roll:mark–end to group a series of operations as one single a single operation for rollback. For example, if a procedure is written to create a rectangle by creating four lines, a rollback can be inserted to undo the entire rectangle as a single state change. Insert the following before the line creation:

```
(roll:mark–start)
```

and the following after creation:

```
(roll:mark–end)
```

The end marks a block of operations that is treated as a single operation for roll back. Roll to mark–start and roll to mark–end blocks may be nested; in which case, the outermost block is treated as a single operation for roll back. This extension returns the delta state number.

Limitations: None

Example:

```
; roll:mark–start
; Mark the beginning of a block of operations.
(roll:mark–start)
;; 0
; Create some test entities.
(define sphere1 (solid:sphere (position 0 0 0) 10))
;; sphere1
(define sphere2 (solid:sphere (position 5 10 20) 10))
;; sphere2
(define sphere3 (solid:sphere (position 20 0 –5) 10))
;; sphere3
; Mark the end of a block of operations.
(roll:mark–end)
;; 0
; Roll back 1 state.
(roll)
;; –1
; All created entities should be gone.
; Roll forward 1 state.
(roll 1)
;; 1
; All created entities should be back.
```

roll:merge–delta–state

Scheme Extension: History and Roll

Action: Merges one delta–state with an adjacent one.

Filename: scm/scmext/roll_scm.cxx

APIs: api_find_named_state

Syntax: (**roll:merge–delta–state** name [stream])

Arg Types:

name	string
stream	history

Returns: string

Errors: None

Description: Merges the named delta state with the adjacent one from the side it rolls to. Prunes history branches, if necessary, to maintain a valid history stream.

name defines the delta–state.

stream specifies the relevant history stream. If is not defined, it is taken from the delta states. If no delta states are provided, it is set to the default stream.

Limitations: None

Example:

```
; roll:merge–delta–state
; create a block
(define b (solid:block (position –10 –10 –10)
    (position 10 10 10)))
;; b
; lop:offset–body corresponds to 1 delta state
(define offset1 (lop:offset–body b –5))
;; offset1
(define offset2 (lop:offset–body b –3))
;; offset2
; Merge the active delta state with the next
; (previous in time) delta state.
(roll:merge–delta–state ”active”)
;; #t
; Since delta states have been merged, rolling back
; one step now is equivalent to rolling back two
; steps before the merge, so the block is restored to
; it’s original size.
(roll)
;; –1
```

roll:merge–delta–states

Scheme Extension: History and Roll

Action: Merges a range of delta states.

Filename: scm/scmext/roll_scm.cxx

APIs: api_merge_states

Syntax:

```
(roll:merge–delta–states [id1 | name1] [id2 | name2]
    [stream] [prune–partners=#f])
```

Arg Types:

id1	integer
id2	integer
name1	string
name2	string
stream	history
prune–partners	boolean

Returns: boolean

Errors: The states do not belong to the same branch or stream, or one of the states is the root delta state.

Description: Merges the range of delta states specified by the id and name arguments (id1, id2, name1, and name2). When names are used, the string “active” may be used to refer to the active delta states. If one of these arguments is omitted, the specified state is merged with its next state. If no names are specified, the active delta state is merged with its predecessor.

stream specifies the relevant history stream. If is not defined, it is taken from the delta states. If no delta states are provided, it is set to the default stream.

By default, the function fails if the range contains states having partner states. However, when prune–partners is set to #t, the branches associated with these partners are pruned.

id1 and id2 are Id’s of the delta state.

name1 and name2 are name of the delta state.

stream specifies the history stream to get the delta state.

Limitations: None

Example:

```
; roll:merge–delta–states
; Create history default.
(define h (history ”default”))
;; h
(define id0 (history:get–active–state–id h))
;; id0
; Create a block.
(define b (solid:block 0 0 0 10 10 10))
;; b
(define id1 (history:get–active–state–id h))
;; id1
; Blend an edge.
(define edges (entity:edges b))
;; edges
(define blend1 (blend:const–rad–on–edge edges 2))
;; blend1
(define id2 (history:get–active–state–id h))
;; id2
(define fix (blend:fix (car edges)))
;; fix
(define id3 (history:get–active–state–id h))
;; id3
; Offset the block.
(define offset (lop:offset–body b 1))
;; offset
(define id4 (history:get–active–state–id h))
;; id4
; Merge the active state with the state immediately
;   following creation of the block.
(define merge (roll:merge–delta–states id4 id1))
;; merge
; Roll takes the part back to the state before block
;   creation.
(roll)
;; –1
```

roll:name–state

Scheme Extension: History and Roll

Action: Attaches a name to the current state for rolling.

Filename: scm/scmext/roll_scm.cxx

APIs: api_pm_name_state

Syntax: `(roll:name–state name–string [history])`

Arg Types:

name–string	string
history	history

Returns: string

Errors: None

Description: name–string identifies the name to attach to the current state. This extension returns the input name–string.

The optional history is used to find the history stream for naming the current state. The default is the stream associated with the active part.

Limitations: None

Example:

```
; roll:name–state
; Create some test entities.
(define sphere1 (solid:sphere (position 0 0 0) 10))
;; sphere1
(define sphere2 (solid:sphere (position 5 10 20) 10))
;; sphere2
; Name the current state.
(roll:name–state ”clock_cr”)
;; ”clock_cr”
(define sphere3 (solid:sphere (position 20 0 –5) 10))
;; sphere3
; Roll back to the start of this session.
(roll ”start”)
;; 6
; All created entities should be gone.
; Roll forward to the named state.
(roll ”clock_cr”)
;; 5
; Two of the created entities should be back.
```

roll:named–states

Scheme Extension: History and Roll

Action: Gets a list of states matching a wildcard name.

Filename: scm/scmext/roll_scm.cxx

APIs: None

Syntax: `(roll:named–states name–pattern [history])`

Arg Types:

name–pattern	string
history	history

Returns: (string ...)

Errors: None

Description: name–pattern identifies the name to attach to the current state.

The optional history is used to find the history stream for naming the current state. The default is the stream associated with the active part.

Limitations: None

Example:

```
; roll:named–states
; Create some test entities.
(define sphere1 (solid:sphere (position 0 0 0) 10))
;; sphere1
(define sphere2 (solid:sphere (position 5 10 20) 10))
;; sphere2
; Name the current state.
(roll:name–state ”clock_cr”)
;; ”clock_cr”
(define sphere3 (solid:sphere (position 20 0 –5) 10))
;; sphere3
; Roll back to the start of this session.
(roll ”start”)
;; 6
; List all states with names in the stream
; associated with the active part.
(roll:named–states ”*”)
;; (”clock_cr” ”begin”)
; roll:set–max–states
; Save only the previous 10 states for undo.
(roll:set–max–states 10)
;; 0
```

roll:set–logging

Scheme Extension: History and Roll

Action: Sets the global logging flag.

Filename: scm/scmext/roll_scm.cxx

APIs: None

Syntax: (**roll:set–logging** on–off)

Arg Types: on–off real

Returns: boolean

Errors: None

Description: Sets the global logging flag which controls the creation of bulletin boards. When logging is turned off, the ability to recover from errors by rolling the model to the last stable state is also turned off.

This routine is intended only for testing and learning about the behavior of the modeler.

Limitations: None

Example:

```
; roll:set–logging
; logging defaults to #t (on)
; verify logging is on.
(roll:get–logging)
;; #t
; turn logging off
(roll:set–logging #f)
;; #t
; Create a solid block.
(define block1 (solid:block (position 0 0 0)
     (position 20 20 20)))
;; block1
; set the color of block1 to blue
(entity:set–color block1 3)
;; ()
; attempt a roll command to return block1 to it's
; original color.
(roll)
;; –1
; the roll did not occur.
```

```
; clear out all entities
(part:clear)
;; #t
; turn logging on
(roll:set–logging #t)
;; #f
; Redefine the solid block1
(define block1 (solid:block (position 0 0 0)
     (position 20 20 20)))
;; block1
; set the color of block1
(entity:set–color block1 3)
;; ()

; attempt a roll command to return block1 to it's
; original color.
(roll)
;; –1
; You should be able to tell the roll completed
; successfully by the block reverting to it's
; original color.
(roll:set–logging #f)
;; #t
```

roll:set–max–states

Scheme Extension: History and Roll

Action: Sets the maximum number of delta states to keep.

Filename: scm/scmext/roll_scm.cxx

APIs: None

Syntax: (**roll:set–max–states** num [history])

Arg Types:		
	num	integer
	history	history

Returns: integer

Errors: None

Description: By default, the system maintains all delta states until the states are explicitly deleted or the system runs out of memory. This extension sets the number of states to keep. By setting this to a small number, the system uses less memory, but it is not able to roll back or forward as many operations.

num sets the number of states to keep. If the extension is called with a num of 0, no delta states are kept, and the undo capability is disabled. If num is –1, the system saves as many states as it can.

The optional history is used to find the history stream on which to set the limit. The default is the stream associated with the active part.

This extension returns the number of states deleted (if any) to satisfy the new limit.

Limitations: None

Example:

```
; roll:set–max–states
; turn on distributed history
(part:set–distribution–mode #t)
;; #t
(roll:set–max–states 1)
;; 0
; state should be empty
(part:entities)
;; ()
; create a block
(define block1 (solid:block 0 0 0 10 10 10))
;; block1
; state should have 1 block
(part:entities)
;; (#[entity 1 1])
; create second block
(define block2 (solid:block 20 20 20 30 25 22))
;; block2
; state should have 2 blocks
(part:entities)
;; (#[entity 1 1] #[entity 2 1])
; attempt to roll back one state
(roll –1)
;; –1
; roll forward to state with 2 blocks
(roll +1)
;; 1
; attempt to roll back two states
(roll –2)
;; –1
; only rolled back one state
; should still be one block
(part:entities)
;; (#[entity 1 1])
```

roll:to–state–id

Scheme Extension: History and Roll

Action: Rolls to a delta state specified by its integer identifier.

Filename: scm/scmext/roll_scm.cxx

APIs: api_change_to_state, api_get_state_from_id, api_note_state

Syntax: (**roll:to–state–id** state–id [history])

Arg Types:

state–id	integer
history	history

Returns: integer

Errors: None

Description: The required state–id variable specifies the integer value associated with a given state. The target state can be specified anywhere in the history stream.

The optional history is used to find the history stream to roll. The default is the stream associated with the active part.

This extension returns the actual number of steps rolled.

Limitations: None

Example:

```
; roll:to-state-id
; turn on distributed history
(part:set-distribution-mode #t)
;; #t
; create a block
(define block1 (solid:block 0 0 0 10 10 10))
;; block1
(define id (history:get-active-state-id))
; id
; state should have 1 block
(part:entities)
;; (#[entity 1 1])
; create second block
(define block2 (solid:block 20 20 20 30 25 22))
;; block2
; state should have 2 blocks
(part:entities)
;; (#[entity 1 1] #[entity 2 1])
; attempt to roll back one state
(roll:to-state-id id)
;; 1
; should be one block
(part:entities)
;; (#[entity 1 1])
```

system:bell

Scheme Extension: Scheme AIDE Application

Action: Sounds the system bell at a specified frequency for a specified duration.

Filename: scm/scmext/sys_scm.cxx

APIs: None

Syntax:

```
(system:bell [duration=hardware-settings]
    [frequency=hardware-settings])
```

Arg Types:		
	duration	integer
	frequency	integer

Returns: unspecified

Errors: None

Description: Set the optional duration in milliseconds.

Set the optional frequency in hertz. When this extension executes, it sounds the system bell or alarm based on the settings.

Note *Some hardware does not include these functions.*

Limitations: None

Example:
```
; system:bell
; Sound the system bell.
(system:bell 5000 5000)
;; ()
(system:bell)
;; ()
```

system:command

Scheme Extension: Scheme AIDE Application

Action: Executes a system command.

Filename: scm/scmext/sys_scm.cxx

APIs: None

Syntax: (**system:command** cmd)

Arg Types: cmd string

Returns: string

Errors: None

Description: This extension executes a system command from the Scheme interpreter command window. The command results, such as the file list from the ls command shown in the following example, displays in the UNIX window.

Limitations: This extension is intended for use on UNIX systems.

Example:
```
; system:command
; Display a file list in the UNIX window.
(system:command "ls /")
;; 0
```

system:getenv

Scheme Extension: Scheme AIDE Application

Action: Gets the value of an environment variable.

Filename: scm/scmext/sys_scm.cxx

APIs: None

Syntax: (**system:getenv** name–string)

Arg Types: name–string string

Returns: string | boolean

Errors: None

Description: The argument name–string specifies the name of the environment variable. This extension returns the value of the environment variable if it is found; otherwise, it returns #f.

Limitations: None

Example:
```
; system:getenv
; Get the value of UNIX environment variable.
(system:getenv ”DISPLAY”)
;; #f
```

system:name

Scheme Extension: Scheme AIDE Application

Action: Gets a string specifying the hardware environment.

Filename: scm/scmext/sys_scm.cxx

APIs: None

Syntax: (**system:name**)

Arg Types: None

Returns: string

Errors: None

Description: Refer to Action.

Limitations: None

Example:
```
; system:name
; Get the system hardware type name.
(system:name)
;; ”hp700”
```

system:play–sound

Scheme Extension: Scheme AIDE Application

Action: Plays a .wav file on Windows.

Filename: scm/scmext/sys_scm.cxx

APIs: None

Syntax: `(system:play–sound filename [async=1])`

Arg Types:

filename	string | boolean
async	integer

Returns: boolean

Errors: None

Description: This extension plays a .wav file. It is only available in the WIN32 environments. The filename is the name of a .wav file. If the first argument is #f, then the system will stop playing any sound that is currently playing.

The optional async argument controls how the sound is played. Valid values for the second argument are:

0 = Play the sound synchronously; i.e., wait for the sound to finish before returning.
1 = Play the sound asynchronously (the default).
2 = Play the sound asynchronously in a loop.

Limitations: NT platforms only.

Example:

```
; system:play–sound
; Plays a sound file
(system:play–sound "chord.wav")
;; #t
```

system:set–timer–off

Scheme Extension: Scheme AIDE Application

Action: Sets the timer off.

Filename: scm/scmext/sys_scm.cxx

APIs: None

Syntax: `(system:set–timer–off)`

Arg Types: None

Returns: boolean

Errors: None

Description: Refer to Action.

Limitations: None

Example:
```
; system:set–timer–off
; Set the timer for 10 seconds then turn it off
(system:set–timer–on 10)
;; #t
(system:set–timer–off)
;; #t
```

system:set–timer–on

Scheme Extension: Scheme AIDE Application

Action: Sets the timer on.

Filename: scm/scmext/sys_scm.cxx

APIs: None

Syntax: `(system:set–timer–on num–sec)`

Arg Types: num–sec integer

Returns: boolean

Errors: None

Description: This extension interrupts an active procedure after a given number of seconds. On UNIX platforms only the virtual time counts, while on NT the timer counts the real time.

num–sec gives the number of seconds after which an active procedure is interrupted.

Limitations: None

Example:
```
; system:set–timer–on
; Set the timer for 10 seconds
(system:set–timer–on 10)
;; #t
```

system:sleep

Scheme Extension: Scheme AIDE Application

Action: Freezes the view for a specified amount of time.

Filename: scm/scmext/sys_scm.cxx

APIs: None

Syntax: (**system:sleep** duration)

Arg Types: duration integer

Returns: integer

Errors: None

Description: This extension freezes a view of an object during journal file processing or loading. Set duration in milliseconds, which the extension returns.

Limitations: None

Example:

```
; system:sleep
; Create solid block 1.
(define block (solid:block
    (position 0 0 0) (position 10 10 10)))
;; block
; Execute a system sleep.
(system:sleep 5000)
;; 5000
; The system pauses.
; After the sleep time expires, create solid block 2.
(define block2 (solid:block
    (position 5 5 5) (position 15 15 15)))
;; block2
```

system:time–string

Scheme Extension: Scheme AIDE Application

Action: Gets a string specifying the current system time and date.

Filename: scm/scmext/sys_scm.cxx

APIs: None

Syntax: (**system:time–string**)

Arg Types: None

Returns: string

Errors: None

Description: This extension returns the system time string as day, month, date, time, and year.

Limitations: None

Example:
```
; system:time–string
; Get the day, month, date, time, and year.
(system:time–string)
;; "Mon Apr 7 10:46:43 1997"
```

ui:error–dialog

Scheme Extension: Scheme AIDE Application

Action: Displays a message string in an error dialog box.

Filename: scm/scmext/ui_scm.cxx

APIs: None

Syntax:
```
(ui:error–dialog msg–string [x y
    width=string–dependent  height=107])
```

Arg Types:

msg–string	string
x	integer
y	integer
width	integer
height	integer

Returns: boolean

Errors: None

Description: Displays a message string in an error dialog box.

msg–string specifies the desired output error message alerting the user of a specific problem.

The optional arguments x, y, width, and height specify the location and size of the dialog box on UNIX. NT ignores these arguments.

This extension returns #f.

Limitations: None

Example:
```
; ui:error–dialog
; Create a user interface error dialog box.
(ui:error–dialog
    ”This function cannot be completed.”)
; Select the OK button.
;; #f
```

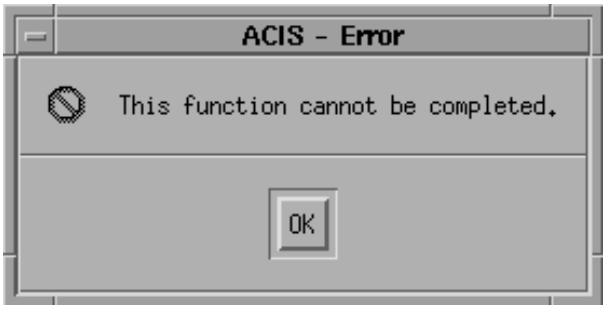

Figure 4-9. ui:error–dialog

ui:info–dialog

Scheme Extension: Scheme AIDE Application

Action: Displays a message string in an information dialog box.

Filename: scm/scmext/ui_scm.cxx

APIs: None

Syntax:
```
(ui:info–dialog msg–string [x y
    width=string–dependent  height=111])
```

Arg Types:		
	msg–string	string
	x	integer
	y	integer
	width	integer
	height	integer

Returns: boolean

Errors: None

Description: Information messages indicate to the user that a process is occurring, the status of a process, etc.

msg–string specifies the desired output message informing the user of the status of a process. x

The optional x, y, width, and height arguments specify the location and size of the dialog box on UNIX. NT ignores these arguments.

This extension returns #t.

Limitations: None

Example:

```
; ui:info–dialog
;; Create a user interface information dialog box.
(ui:info–dialog
    "Command is being processed. Please Wait.")
; Select the OK button.
;; #t
```

Figure 4-10. ui:info–dialog

ui:prompt

Scheme Extension: Scheme AIDE Application

Action: Displays a message string on the command line.

Filename: scm/scmext/ui_scm.cxx

APIs: None

Syntax: (**ui:prompt** msg–string)

Arg Types:	msg–string	string

Returns: unspecified

Errors: None

Description: Displays a message string on the command line. msg–string specifies the desired output message. Typically, it instructs the user to supply additional information to complete a process.

On Windows, this command sends the msg–string via DDE to the server application. On other systems, it sends the string to stdout, which defaults to the Scheme command window.

Limitations: None

Example:
```
; ui:prompt
; Create a user interface prompt on the command line.
(ui:prompt "Select two positions.")
;; Select two positions.
;; ()
```

ui:warning–dialog

Scheme Extension: Scheme AIDE Application

Action: Displays a message string in a warning dialog box.

Filename: scm/scmext/ui_scm.cxx

APIs: None

Syntax:
```
(ui:warning–dialog msg–string [x y
    width=string–dependent  height=109])
```

Arg Types:		
	msg–string	string
	x	integer
	y	integer
	width	integer
	height	integer

Returns: boolean

Errors: None

Description: Displays a message string in a warning dialog box.

msg–string specifies the output message that warns the user that executing the next process may retrieve unintentional results. The user then has the option to continue or cancel.

The optional x, y, width, and height arguments specify the location and size of the dialog box on UNIX. NT ignores these arguments.

Limitations: None

Example:
```
; ui:warning–dialog
; Create a user interface warning dialog box.
(ui:warning–dialog
    ”If you continue the data may be lost.”)
; Select the Cancel button.
;; #f
```

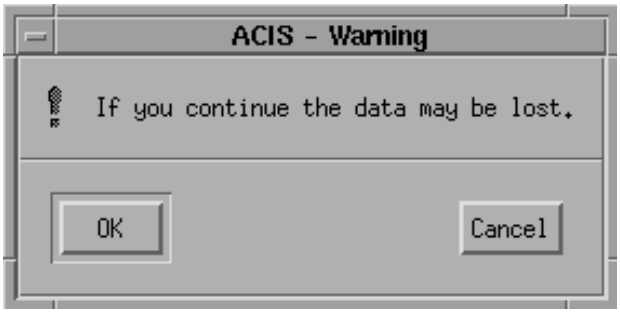

Figure 4-11. ui:warning–dialog

ui:yesno–dialog

Scheme Extension: Scheme AIDE Application

Action: Displays a message string in a yes/no dialog box.

Filename: scm/scmext/ui_scm.cxx

APIs: None

Syntax:
```
(ui:yesno–dialog msg–string [title–string=ACIS]
    [x y width=string–dependent height=109])
```

Arg Types:		
	msg–string	string
	title–string	string
	x	integer
	y	integer
	width	integer
	height	integer

Returns: boolean

Errors: None

Description: Displays a message string in a yes/no dialog box.

msg–string specifies the output message dialog. This extension uses yes/no to query the user to complete some function.

The optional title–string specifies the heading for a specific set of messages.

The optional x, y, width, and height specify the location and size of the dialog box on UNIX. NT ignores these arguments.

Limitations: None

Example:

```
; ui:yesno–dialog
; Create a user interface yes/no dialog box.
(ui:yesno–dialog "Do you want to save the part?"
    "ACIS for More Fun")
; Select the No button.
;; #f
```

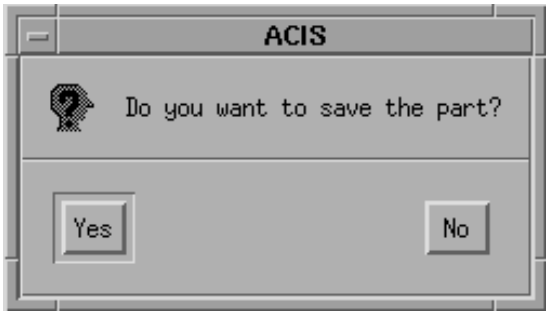

Figure 4-12. ui:yesno–dialog

versionid

Scheme Extension: Scheme AIDE Application

Action: Gets the current version ID of the ACIS executable.

Filename: scm/scmext/ver_scm.cxx

APIs: None

Syntax: (**versionid**)

Arg Types: None

Returns: string

Errors: None

Description: Refer to Action.

Limitations: None

Example:
```
; versionid
; Get the current version of the executable.
(versionid)
;; "Acis version 8.0 (ag1.51): for HP-UX B.10.20
;; generated Fri Mar  1 15:09:00 2002"
```

view:bg–color

Scheme Extension: Viewing, Backgrounds and Foregrounds, Colors

Action: Gets a view's background color.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:bg–color** [view=active])

Arg Types: view — view

Returns: color

Errors: None

Description: The optional argument view determines from which view to get the background. The default background colors of views is black RGB = 0,0,0).

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:
```
; view:bg–color
; Define a view.
(define view1 (view:dl 0 0 100 100))
;; view1
; Get the background color of a view.
(view:bg–color view1)
;; #[color 0 0 0]
```

view:clear

Scheme Extension: Viewing

Action: Clears the contents of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:clear** [view=active])

Arg Types: view view

Returns: view

Errors: None

Description: Cleans all drawings from the specified window. The objects depicted are not destroyed. The visual representation may be refreshed using view:refresh. If the view is associated with a file, a “new page” instruction is written to the file.

The argument view is optional. It specifies the view to clear. If view is not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:clear
; Define a named view.
(define view (view:dl))
;; view
; Create a solid block that appears in both views.
(define block1
    (solid:block (position –10 –10 –10)
    (position 5 10 15)))
;; block1
; Clear the active view.
(view:clear)
;; #[view 1075533160]
; Clear the named view.
(view:clear view)
;; #[view 1075519936]
```

view:delete

Scheme Extension: Viewing

Action: Deletes a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:delete [view=active])`

Arg Types: view view

Returns: unspecified

Errors: None

Description: Deletes the specified view. This involves not only removing the associated window from the computer screen, but also deleting all mention of the view from the list returned by env:views.

The argument view is optional. It specifies the view to clear. If view is not specified, the active view is deleted. The view has to have been created using a define statement. If the view is associated with a file, it writes any buffered information to the file, closes it, and then deletes the view from the list. Deleting a view does not destroy the objects represented in it.

This command expects a view that has been defined. If a view has not been defined, its handle number can be used to delete it. For example:

```
(view:delete (view:with–handle 1075892672))
```

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:
```
; view:delete
; Create a new view.
(define view1 (view:dl))
;; view1
; Delete the new view.
(view:delete view1)
;; ()
```

view:display–facets

Scheme Extension: Viewing

Action: Displays the facets.

Filename: scm/scmext/pcacis_scm.cxx

APIs: None

Syntax: `(view:display–facets disp)`

Arg Types: disp boolean

Returns: unspecified

Errors: None

Description: Displays the facets when the input value is TRUE.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:display–facets
; open a dl view
(define v (view:dl))
;; v
; create a sphere
(define s (solid:sphere 0 0 0 10))
;; s
; zoom to fill screen
(zoom–all)
;; #[view 2755404]
(view:display–facets #t)
;; facets are shown
(view:display–facets #f)
;; facets are not shown
```

view:display–param–lines

Scheme Extension: Viewing

Action: Displays parameter lines.

Filename: scm/scmext/pcacis_scm.cxx

APIs: None

Syntax: (**view:display–param–lines** disp)

Arg Types: disp boolean

Returns: unspecified

Errors: None

Description: Displays the parameter lines when the input value is TRUE.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:display-param-lines
; open a dl view
(define v (view:dl))
;; v
; create a sphere
(define s (solid:sphere 0 0 0 10))
;; s
; isometric view
(iso)
;; #[view 789352]
; zoom to fill screen
(zoom-all)
;; #[view 789352]
(view:display-param-lines #t)
;; parameter lines are shown
(view:display-param-lines #f)
;; parameter lines are not shown
```

view:display–surface–polys

Scheme Extension: Viewing

Action: Displays surface polygons.

Filename: scm/scmext/pcacis_scm.cxx

APIs: None

Syntax: (**view:display-surface-polys** disp)

Arg Types:	disp	boolean

Returns: unspecified

Errors: None

Description: Displays thesurface polygons when the input value is FALSE.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:display-surface-polys
; open a dl view
(define v (view:dl))
;; v
; create a blended block
(define b (solid:block 0 0 0 10 10 10))
;; b
(define e (car (entity:edges b)))
; isometric view
(iso)
;; #[view 526362]
; zoom to fill screen
(zoom-all)
;; #[view 526362]
;; e
(entity:set-color e RED)
;; ()
;; edge turns red
(define bl (blend:var-rad-on-edge e 2 3))
;; bl
(blend:fix e)
;; #t
(view:display-surface-polys #f)
;; ()
(render:rebuild)
;; ()
;; surface polygons are shown
(view:display-surface-polys #t)
;; ()
(render:rebuild)
;; ()
;;surface polygons are not shown
```

view:draw–point

Scheme Extension: Viewing

Action: Draws a temporary point.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:draw-point** position [view=active])

Arg Types:	position	position
	view	view

Returns: unspecified

Errors: None

Description: Draws a temporary representation of a point in the view. As soon as the view is refreshed, the temporary point representation goes away.

The argument position specifies a location to draw the point.

The optional argument view specifies the view to use for the point. If view is not specified, the active view is used. The style and size of the point representation can be changed using view:set–point–style and view:set–point–size.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:
```
; view:draw–point
; Draw temporary point 1.
(view:draw–point (position 6 6 6))
;; ()
; Draw temporary point 2.
(view:draw–point (position –10 –10 –10))
;; ()
; Clear the view.
(view:refresh)
;; #[view 1075915864]
```

view:draw–polyline

Scheme Extension: Viewing

Action: Draws a temporary polyline.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax:
```
(view:draw–polyline position–list [fill]
    [view=active])
```

Arg Types:		
	position–list	position | (position ...)
	fill	boolean
	view	view

Returns: unspecified

Errors: None

Description: Draws a temporary polyline in the active view or the specified view. As soon as that view is refreshed, polyline representation goes away.

The argument position–list specifies a list of two or more positions to draw the polyline through.

If the optional argument fill is #t, closed polylines are filled with the draw color.

The optional argument view specifies the view in which to draw polyline. If view is not specified, the active view is used. The line style can be changed with the command view:set–line–style; the default is solid.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:draw–polyline
; Draw a temporary polyline to a view.
(define line1
    (view:draw–polyline (list (position 3 3 3)
    (position 6 6 6) (position –10 –10 –10))))
;; line1
; Define a new view.
(define view1
    (view:draw–polyline (list (position 3 3 3)
    (position 6 6 6) (position –10 –10 –10))))
;; view1
; Draw another polyline with fill to a specific view.
(define line2
    (view:draw–polyline (list (position 3 3 3)
    (position –10 –10 –10) (position 40 –10 20)
    #t view1)))
;; line2
; Clear the view.
(view:refresh)
;; #[view 1075907056]
```

view:draw–text

Scheme Extension: Viewing

Action: Draws a temporary text string.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:draw-text position string [view=active])`

Arg Types:

position	position
string	string
view	view

Returns: unspecified

Errors: None

Description: Draws a temporary text string in the specified view or active view. As soon as that view is refreshed, input text goes away.

The argument position specifies the starting location for the text.

The argument string specifies the text to output.

The optional argument view specifies the view in which to draw. If view is not specified, the active view is used. The text is aligned horizontally so that its baseline meets the left edge of the smallest rectangle that fits around it at the given position.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:draw-text
; Draw temporary text to the current view.
(view:draw-text (position 0 0 0) "rectangle 1")
;; ()
; Draw temporary text.
(view:draw-text (position 0 25 0) "sphere")
;; ()
; Clear the view.
(view:refresh)
;; #[view 1075907056]
```

view:eye

Scheme Extension: Viewing

Action: Gets the eye position of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:eye [view=active])`

Arg Types: view view

Returns: position

Errors: None

Description: Retrieves the eye position for the specified view, or the active view.

The optional argument view specifies the view. If view is not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:eye
; Create a block.
(define block1
    (solid:block (position 5 5 5)
    (position 15 15 15)))
;; block1
; Determine the eye point.
(view:eye)
;; #[position 100 –200 100]
; Set a new eye position.
(view:set–eye (position 50 –100 50))
;; #[position 100 –200 100]
; Refresh to see the block
; from the new eye position.
(view:refresh)
;; #[view 1076014304]
; Determine the new eye position.
(view:eye)
;; #[position 50 –100 50]
```

view:fg–color

Scheme Extension: Viewing, Backgrounds and Foregrounds, Colors

Action: Gets a view’s foreground color.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:fg-color [view=active])`

Arg Types: view — view

Returns: color

Errors: None

Description: The optional argument view specifies where to get the foreground color. If no view is specified, the color is retrieved from the active view. Colors are of the form:

```
#[color r g b]
```

where *r*, *g*, and *b* are numerical weights between 0 and 1 for the red, green, and blue color components.

By default, temporary text and graphics are green, so this procedure returns:

```
#[color 0 1 0]
```

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:fg-color
; Define a new view.
(define view1 (view:dl))
;; view1
; Get the foreground color in view.
(view:fg-color view1)
;; #[color 1 1 1]
```

view:handle

Scheme Extension: Viewing

Action: Gets a handle of the specified view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:handle [view=active])`

Arg Types: view — view

Returns: integer

Errors: None

Description: This extension returns the handle for a view window as an integer.

The optional argument view specifies the specific view to get the handle from. If not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:handle
; Define a new view.
(define view1 (view:dl))
;; view1
; Get the window handle for view.
(view:handle view1)
;; 1075958424
; You can delete the view if all you know is
; the window handle.
(view:delete (view:with–handle 1075958424))
;; ()
```

view:height

Scheme Extension: Viewing

Action: Gets the height of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:height** [view=active])

Arg Types: view view

Returns: real

Errors: None

Description: The view height in conjunction with the viewport height determines the scale of the representation. If the view height is doubled, for example, while the dimensions of the viewport remain unchanged, then the images in the view window will shrink to half size.

The argument view specifies the specific view to get the handle from. If not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:
```
; view:height
; Define a new view.
(define view1 (view:dl))
;; view1
; Get the height of view.
(view:height view1)
;; 100
```

view:hither

Scheme Extension: Viewing

Action: Gets the hither clipping distance of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:hither** [view=active])

Arg Types:	view	view

Returns: real

Errors: None

Description: This returns the hither clipping distance. A view's hither plane is a plane perpendicular to the view's line of sight. Objects located on the same side of the hither plane as the eye position are invisible; they are not pictured in the view. The distance along the line of sight from the eye position to its hither plane is the hither clipping distance. The default hither clipping distance is 0.025.

Hither and yon settings are very useful for perspective views or for views where the eye point is inside of the object. Establish clipping such that it clips out everything behind the eye point. It can also be used to clip out foreground and backgrounds objects.

The argument view specifies the specific view to get the handle from. If not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:hither
; Define a new view.
(define view1 (view:dl))
;; view1
; Set the view's hither clip distance.
(view:set-hither .03 view1)
;; #[view 1076057640]
; Get the hither clipping plane for view.
(view:hither view1)
;; 0.03
```

view:out

Scheme Extension: Viewing

Action: Gets the vector from the target to the eye position.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:out** [view=active])

Arg Types: view — view

Returns: gvector

Errors: None

Description: This extension returns the unitized gvector from the view target to the view eye position. This vector is always perpendicular to the screen. However, its relationship to the model is determined by this command.

The argument view specifies the specific view to get the handle from. If not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:out
; Get the out vector for a view.
(view:out)
;; #[gvector 0.408248290463863 -0.816496580927726
;; 0.408248290463863]
```

view:perspective?

Scheme Extension: Viewing

Action: Determines if perspective is on for a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:perspective?** [view=active])

Arg Types: view view

Returns: boolean

Errors: None

Description: Determines if perspective is on for a view. If the output is #t, the view uses the perspective mode; if it is #f, the view uses orthographic projection mode.

The argument view specifies the specific view to get the handle from. If not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:
```
; view:perspective?
; Create a new view called view.
(define view1 (view:dl))
;; view1
; Determine if the current view
; is a perspective view.
(view:perspective?)
;; #f
; Make view a perspective view.
(view:set–perspective #t view1)
;; #f
; Determine if view is a perspective view.
(view:perspective? view1)
;; #t
```

view:right

Scheme Extension: Viewing

Action: Gets the right vector of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:right [view=active])`

Arg Types:

view	view

Returns: gvector

Errors: None

Description: This extension returns the unitized gvector that points toward the right side of the view. This corresponds to the *x*–axis of the view coordinate system. The gvectors returned from the operations view:out, view:right, and view:up form a right–handed orthonormal triple. Each is the cross–product of the other two. The view:out vector is always perpendicular to the screen. After the model has been reoriented, this command determines the *x*–axis relationship to the model.

The argument view specifies the specific view to get the handle from. If not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:right
; Get the right vector for a view.
(view:right)
;; #[gvector 0.894427190999916 0.447213595499958 0]
```

view:set

Scheme Extension: Viewing

Action: Sets a view's eye position, target position, and up vector.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax:

```
(view:set eye–position target–position
    up–direction [view=active])
```

Arg Types:

eye–position	position
target–position	position
up–direction	gvector
view	view

Returns: view

Errors: None

Description: The argument eye–position specifies where one is looking from.

The argument target–position specifies where one is looking to. The eye position must be different from the target.

The argument up–direction specifies the up direction for the view with respect to the model. The gvector extending from the target to the eye position cannot be parallel to the up vector.

The optional argument view specifies the view to set and defaults to the active view. After using the view:refresh extension, objects displayed in the view are redisplayed according to the new view parameters.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set
; Create a block.
(define block1
    (solid:block (position 0 0 0)
    (position 35 35 35)))
;; block1
; Define a new view.
(define view1 (view:dl))
;; view1
; Set a view’s eye position, target position,
; and up vector.
(view:set (position 200 –400 200)
    (position 0 0 0) (gvector 1 0 0) view1)
;; #[view 1075533160]
; Refresh the view.
(view:refresh view1)
;; #[view 1075533160]
```

view:set–bg–color

Scheme Extension: Viewing, Backgrounds and Foregrounds, Colors

Action: Sets a view’s background color.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:set–bg–color** color [view=active])

Arg Types:		
	color	integer | color
	view	view

Returns: view

Errors: None

Description: The argument color specifies the color to set the background, can be as an integer in the range 0–7, or as an rgb color object formed using the color:rgb extension. color integer values are:

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

The optional argument view specifies which view to get a background color. If view is not specified, the active view is used. The specified background color is displayed after calling view:refresh.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–bg–color
; Define a new view.
(define view1 (view:dl))
;; view1
; Set the background color to magenta for the
; view using a color integer.
(view:set–bg–color 6 view1)
;; #[view 1075519376]
; Set the background color to light pink for the
; current view using an rgb value.
(view:set–bg–color (color:rgb 0.9 0.45 0.7))
;; #[view 41943045]
```

view:set–clipping

Scheme Extension: Viewing

Action: Sets the hither and yon clip distances.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:set–clipping** hither=0.025 yon [view=active])

Arg Types:		
	hither	real
	yon	real
	view	view

Returns: view

Errors: None

Description: Specifies the hither clipping distance. Hither specifies the near distance and must be greater than 0. Any item closer to the eye than the hither distance is clipped.

yon specifies the far distance, which must be greater than the hither distance. Any item farther away from the eye than the yon distance is clipped.

The optional argument view specifies the view to set. If view is not specified, the active view is used. A view's hither plane is a plane perpendicular to the view's line of sight. Objects located on the same side of the hither plane as the eye position are invisible; they are not pictured in the view. The distance along the line of sight from the eye position to its hither plane is the hither clipping distance. The default hither clipping distance is 0.025.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–clipping
; Create a solid block for viewing.
(define block1
    (solid:block (position 10 10 10)
    (position –10 40 30)))
;; block1
; OUTPUT Original

; Set a view's hither and yon clipping planes.
(view:set–clipping 10 250)
;; #[view 1075519376]
(view:refresh)
;; #[view 1075915640]
; OUTPUT Result
```

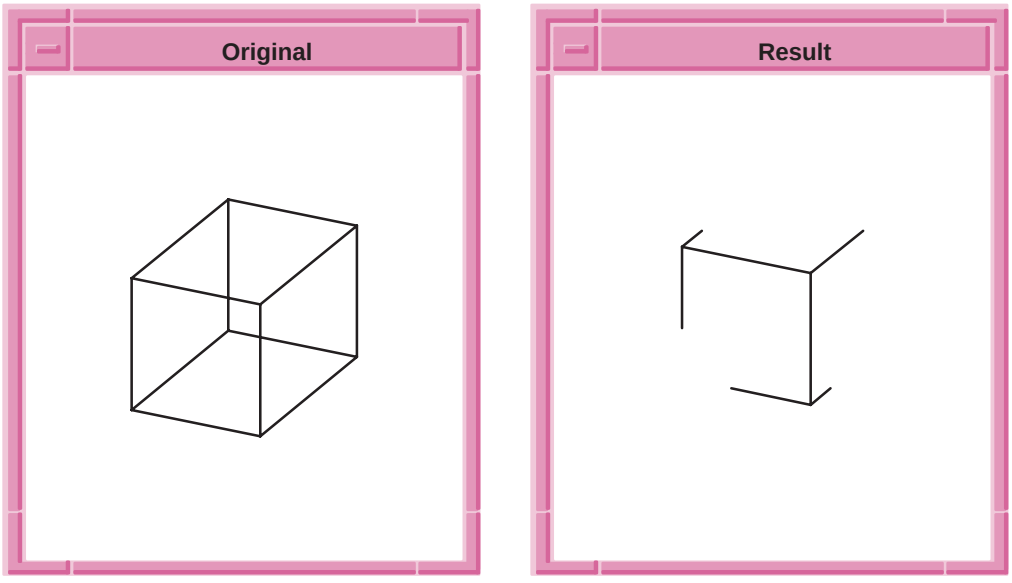

Figure 4-13. view:set–clipping

view:set–draw–mode

Scheme Extension: Viewing

Action: Sets the drawing mode for temporary graphics in a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:set–draw–mode mode [view=active])`

Arg Types:		
	mode	integer
	view	view

Returns: view

Errors: None

Description: Sets the drawing mode for the wireframe representation of entities and the depiction of temporary points and polylines. Valid drawing modes include:

[mode = 1]	Normal drawing mode, or copy mode. Wireframe uses color specified in env:default–color and the temporary elements use color specified by view:fg–color.
[mode = 2]	XOR (exclusive or) mode. Each pixel is colored based an exclusive or logical combination of previous drawing color and wireframe env:default–color and view:fg–color temporary colors.

Temporary text is always displayed in copy mode.

The optional argument view specifies the view to set the drawing mode. If view is not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–draw–mode
; Define a new view.
(define view1 (view:dl))
;; view1
; Create a solid block.
(define block1
    (solid:block (position –10 –5 0)
    (position 10 8 6)))
;; block1
; Create a polyline to see the difference.
(view:draw–polyline (list (position –10 –10 –10)
    (position 2 2 2) (position 10 2 2)
    (position –10 2 2)) #t view1)
;; ()
; Set the drawing mode of a view to exclusive or.
(view:set–draw–mode 2 view1)
;; #[view 1075519376]
; Refresh the view.
(view:refresh view1)
;; #[view 1075533160]
(view:set–draw–mode 1 view1)
;; #[view 1075519376]
; Refresh the view.
(view:refresh view1)
;; #[view 1075533160]
```

view:set–eye

Scheme Extension: Viewing

Action: Sets a view's eye position.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:set–eye** eye–position=(0,0,0) [view=active])

Arg Types:	eye–position	position
	view	view

Returns: position

Errors: None

Description: The argument eye–position specifies the eye location of the view. It sequentially resets the values of view:out, view:right, and view:up according to new view.

The optional argument view specifies the view to set, and defaults to the active view. This extension returns the previous eye position. view:refresh is required to see changes.

The eye position needs to be different from the target position. The resulting vector from the target to the eye position cannot be parallel to the up gvector.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–eye
; Define a new view.
(define view1 (view:dl))
;; view1
; Create a solid block.
(define block1
    (solid:block (position –10 –5 0)
    (position 10 8 6)))
;; block1
; Set the eye position of the new view.
(view:set–eye (position 15 –20 0) view1)
;; #[position 0 0 500]
; Refresh the view.
(view:refresh view1)
;; #[view 1075533160]
; Get the eye position of the new view.
(view:eye view1)
;; #[position 15 –20 0]
```

view:set–fg–color

Scheme Extension: Viewing, Backgrounds and Foregrounds, Colors

Action: Sets the foreground color for a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:set–fg–color** color [view=active])

Arg Types:		
	color	color
	view	view

Returns: view

Errors: None

Description: Sets the foreground color for a view. The argument color specifies the color to set the foreground, can be an integer in the range 0–7, or an rgb color object formed using the color:rgb extension. color values include:

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

The optional argument view specifies which view to set the foreground color. If view is not specified, the active view is used. The specified foreground color is displayed after calling view:refresh.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–fg–color
; Define a new view.
(define view1 (view:dl))
;; view1
; Set the foreground color to magenta.
(view:set–fg–color 6 view1)
;; #[view 1075519376]
; Create something to be displayed in color.
(view:draw–text (position 0 0 0) ”howdy” view1)
;; ()
```

view:set–font

Scheme Extension: Viewing

Action: Sets the text font for a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:set–font font–string [view=active])`

Arg Types:

font–string	string
view	view

Returns: view

Errors: Font is unsupported.

Description: Sets the text font for a view. The argument font–string specifies the type of text font to use in a view.

The optional argument view specifies the view to set the font, and defaults to the active view.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–font
; Define a new view.
(define view1 (view:dl))
;; view1
; Set the text font for a view.
(view:set–font ”helvetica” view1)
;; #[view 1075533160]
; Draw text in that font.
(view:draw–text (position 0 0 0)
    ”helvetica font appears” view1)
;; ()
```

view:set–font–size

Scheme Extension: Viewing

Action: Sets the text font size for a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:set–font–size size [view=active])`

Arg Types:

size	real
view	view

Returns: view

Errors: None

Description: Sets the text font size for a view. size specifies how large to make the text font.

The optional view specifies the view to set the font size. If view is not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–font–size
; Define a new view.
(define view (view:dl))
;; view
; Sets the text font size for a view.
(view:set–font–size 14 view)
;; #[view 1075519376]
; Draw text in that font.
(view:draw–text (position 0 0 0)
    ″font size 14″ view)
;; ()
```

view:set–hither

Scheme Extension: Viewing

Action: Sets a view’s hither clip distance.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:set–hither hither=0.025 [view=active])`

Arg Types:

hither	real
view	view

Returns: view

Errors: None

Description: This extension sets the hither clip distance from the active view unless the optional argument view is specified. A view's hither plane is a plane perpendicular to the view's line of sight. Objects located on the same side of the hither plane as the eye position are invisible; they are not pictured in the view. The distance along the line of sight from the eye position to its hither plane is the hither clipping distance. The default hither clipping distance is 0.025.

Hither and yon settings are very useful for perspective views or for views where the eye point is inside of the object. Establish clipping such that it clips out everything behind the eye point. It can also be used to clip out foreground and backgrounds objects.

Specifies the hither clipping distance. Hither specifies the near distance and must be greater than 0.

The optional view specifies the view to set the font size. If view is not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–hither
; Get the hither clip distance.
(view:hither)
;; 0.025
(define block1
    (solid:block (position –30 –30 –30)
    (position 20 50 20)))
;; block1
; OUTPUT Original

; Set the hither clip distance.
(view:set–hither 220)
;; #[view 1075519376]
(view:refresh)
;; #[view 1075519376]
; OUTPUT Result
```

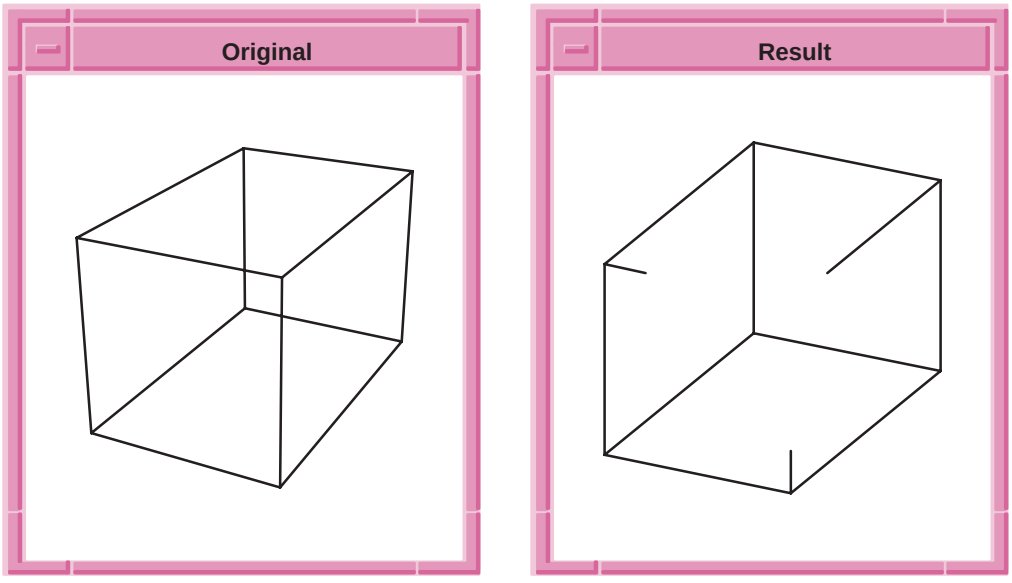

Figure 4-14. view:set–hither

view:set–line–style

Scheme Extension: Viewing

Action: Sets the line style for temporary lines.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:set-line-style style-string [view=active])`

Arg Types:	style–string	string
	view	view

Returns: view

Errors: None

Description: Sets the line style for temporary lines in a view. The argument style–string (string) specifies the style of lines displayed in a view. Valid settings include:

“.” (period) used for dotted lines
“–” (dash) used for dashed lines
“_” (underline) used for solid lines

The optional argument view specifies the view for setting the line style. If view is not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–line–style
; Define a new view.
(define view1 (view:dl))
;; view1
; Set the line style for view.
(view:set–line–style ”.” view1)
;; #[view 1075419376]
(view:draw–polyline (list
    (position 0 0 0) (position 10 10 10)) view1)
;; ()
```

view:set–line–width

Scheme Extension: Viewing

Action: Sets the width of temporary lines in a view created with view:draw–polyline.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: **(view:set–line–width** size=1 [view=active])

Arg Types:		
	size	integer
	view	view

Returns: view

Errors: None

Description: Sets the width of temporary lines in a view. The argument size specifies how wide to make the line. The default value of line width is 1. On the Windows platform, if size is greater than 1 and a nonsolid line style is used, then the line style is temporarily overridden. In this case, the solid line style is used until the size is set back to 1.

The optional argument view specifies the view that sets the line size. If view is not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–line–width
; Set the size of temporary lines for a view.
(view:set–line–width 5)
;; #[view 1075519376]
(view:draw–polyline (list
    (position 0 0 0) (position 10 10 10)))
;; ()
```

view:set–perspective

Scheme Extension: Viewing

Action: Sets the perspective or orthographic modes.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:set–perspective** persp–orth [view=active])

Arg Types:		
	persp–orth	boolean
	view	view

Returns: boolean

Errors: None

Description: Sets the perspective of a view to perspective or orthographic. The argument persp–orth toggles between perspective mode (#t) and orthographic mode (#f). view

The optional argument view specifies view that sets the perspective. If view is not specified, the active view is used. This extension returns the previous state. To be displayed in the view, a view:refresh is required.

Hither and yon settings are very useful for perspective views or for views where the eye point is inside of the object. Establish clipping such that it clips out everything behind the eye point. It can also be used to clip out foreground and backgrounds objects.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–perspective
; Determine if perspective is on.
(view:perspective?)
;; #f
; Create a block.
(define block1
    (solid:block (position –30 –30 –30)
    (position 20 50 20)))
;; block1
; OUTPUT Original

; Set the perspective mode on for a view.
(view:set–perspective #t)
;; #f
(view:refresh)
;; #[view 1076139208]
; Set the orthographic mode on for a view.
(view:set–perspective #f)
;; #t
(view:refresh)
;; #[view 1076139208]
; OUTPUT Result
```

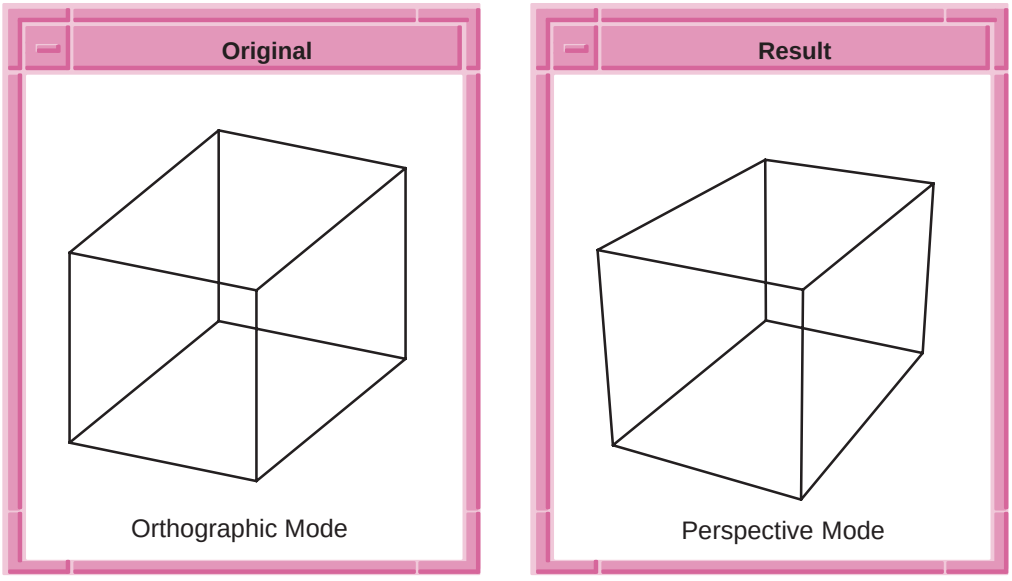

Figure 4-15. view:set–perspective

view:set–point–size

Scheme Extension: Viewing

Action: Sets the size of temporary points in a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:set–point–size size=10 [view=active])`

Arg Types:

size	integer
view	view

Returns: view

Errors: None

Description: The argument size specifies how large to make the point. The default size is 10.

The optional argument view specifies the view that sets the point size. If view is not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–point–size
; Set the point style.
(view:set–point–style "o")
;; #[view 1075533160]
; Draw a point.
(view:draw–point (position 3 6 8))
;; ()
; Set the size of temporary points.
(view:set–point–size 20)
;; #[view 1075519376]
; Draw a second point in the same position.
(view:draw–point (position 3 6 8))
;; ()
```

view:set–point–style

Scheme Extension: Viewing

Action: Sets the style for temporary points in a view created with view:draw–point.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:set–point–style** style–string [view=active])

Arg Types:

style–string	string
view	view

Returns: view

Errors: None

Description: Sets the style for temporary points in a view. The argument style–string specifies the style of points displayed in a view. Valid settings include:

x, display point as x
X, display point as X
+, display point as + (plus)
., display point as . (period)
o, display point as o
O, display point as O

The optional argument view specifies the view for the point style. If view is not specified, the active view is used.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–point–style
; Set the style for temporary points.
(view:set–point–style "+")
;; #[view 1075533160]
; Draw a point.
(view:draw–point (position 3 6 8))
;; ()
```

view:set–rb–mode

Scheme Extension: Viewing, Rubberbanding

Action: Sets the rubberbanding mode for a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(`**`view:set–rb–mode`** `on–off [view=active])`

Arg Types:

on–off	boolean
view	view

Returns: view

Errors: None

Description: Sets the rubberbanding mode for a view. The argument on–off turns rubberbanding on (#t) or off (#f).

The optional argument view specifies the view that uses rubberbanding. If the view is associated with a window, it turns on the XOR drawing mode used while rubberbanding wireframe representations. If the view is associated with a file, this command has no effect. If view is not specified, the active view is used.

This is used in the drawing and undrawing routines which are built into the Scheme rubberband drivers. These drivers have user–configurable information relating to effects when a mouse enters a window, moves within a window, or leaves a window. It also handles what happens when the window is repainted while the mouse cursor is in it.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–rb–mode
; Define a new view.
(define view (view:dl))
;; view
; Set the rubberbanding mode for view.
(view:set–rb–mode #f view)
;; #[view 1075519376]
```

```
; Another example.
(define rbd:line2
    (lambda (boo1 . init-pos)
        (let* ((pos1 (if (null? init-pos)
            (wcs:origin)
            (car init-pos)))
        (pos2 #f)
        (draw
        (lambda ()
            (view:set-rb-mode #t)
            (view:draw-polyline
            (list pos1 pos2))
            (view:set-rb-mode #f))))
        (rbd:scheme
        bool
        (vector
        ; init-hook
        ' ()
        ; start-hook
        (lambda (self evt)
            (set! pos2 (pick:position evt))
        (draw))
        ; update-hook
        (lambda (self evt)
            (draw))
            (set! pos2 (pick:position evt))
            (draw))
        ; stop-hook
        (lambda (self) (draw))
        ; repaint-hook
        (lambda (self view) (draw))
        ; position-hook
        '()
        ; end-hook
        '())))))
;; rbd:line2
```

view:set–size

Scheme Extension: Viewing

Action: Sets the width and height of the view window in world coordinates.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:set-size** width height [view=active])

Arg Types:

width	real
height	real
view	view

Returns: view

Errors: None

Description: Resets the width and height of the view window in world coordinates. The argument width specifies the horizontal size of the view window.

The argument height specifies the vertical size of the window. The width and height specified may be adjusted to fit the actual aspect ratio of the view window on the screen. Whichever dimension value has a greater ratio to its current value is reset. The other dimension is set to the value required to prevent image from being stretched or shrunk. To be visible, requires a view:refresh. view

The optional view specifies the view to set, and it defaults to the active view.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set-size
; Define a WCS.
(define wcs (wcs (position 0 0 0) (gvector 1 0 0)
    (gvector 0 1 0)))
;; wcs
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 35 35 35)))
;; block1
; Set the size of the view.
(view:set-size 300 500)
;; #[view 1075533160]
; Refresh the view.
(view:refresh)
;; #[view 41943045]
; OUTPUT Original
```

```
; Set the size of the view.
(view:set-size 100 200)
;; #[view 1075533160]
; Refresh the view.
(view:refresh)
;; #[view 41943045]
; OUTPUT Result
```

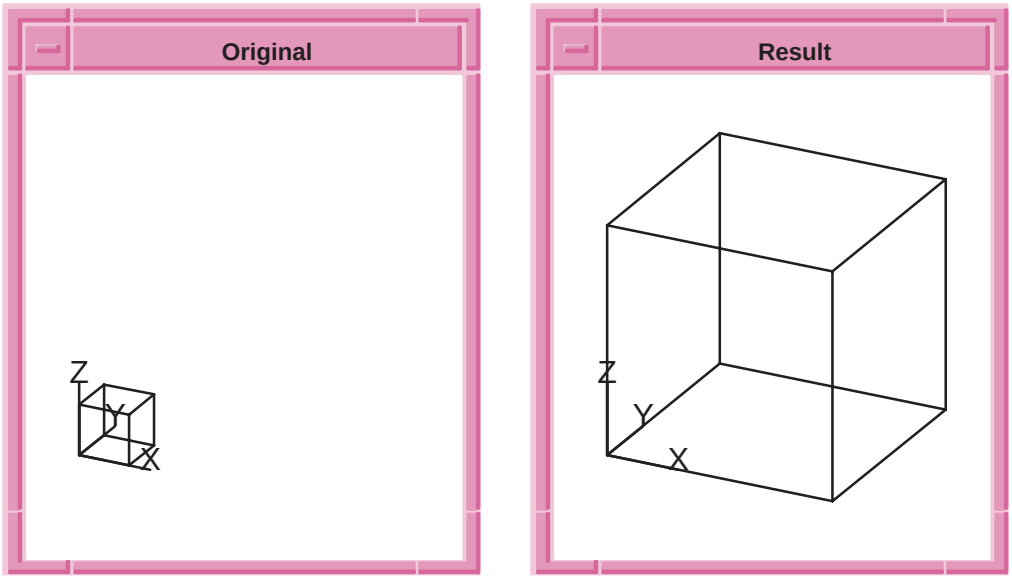

Figure 4-16. view:set–size

```
; Another example where the size of the view
; and viewport are changed before an output image
; is made.
; Define a new view.
(define view2 (view:dl 100 200))
;; view2
; This queries the current view size. Then
; it resizes the viewport. Then it makes the
; object appear smaller by changing the width
; and height parameter of the view.
(let ((w (view:width view2)))
    (view:set–viewport 0 0 300 300 view2)
    (view:set–size (* w 2) (* w 2) view2))
;; #[view 109820984]
; Refresh the view to see the changes
(view:refresh view2)
;; #[view 109829897]
```

view:set–target

Scheme Extension: Viewing

Action: Sets a view's target position.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax:
```
(view:set–target target–position=(0,0,0)
    [view=active])
```

Arg Types:		
	target–position	position
	view	view

Returns: position

Errors: None

Description: The argument target–position specifies the target location of the view. It sequentially resets the values of view:out, view:right, and view:up according to new view.

The target position needs to be different from the eye position. The resulting vector from the target to the eye position cannot be parallel to the up gvector.

The optional argument view specifies the view to set, and defaults to the active view. This extension returns the previous eye position. view:refresh is required to see changes.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–target
; Define a new view.
(define view (view:dl))
;; view
(view:target view)
;; #[position 0 0 0]
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 20 20 20)))
;; block1
; Set the target position for a view.
(view:set–target (position 0 40 0) view)
;; #[position 0 0 0]
; Refresh the view.
(view:refresh)
;; #[view 1075533160]
; Get the target position of the new view.
(view:target view)
;; #[position 0 40 0]
```

view:set–title

Scheme Extension: Viewing

Action: Sets the banner name for a specified view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:set–title** name–string [view=active])

Arg Types:		
	name–string	string
	view	view

Returns: view

Errors: None

Description: Sets the banner name for a specified view. The argument name–string specifies the name to place in the banner. It is generally in double quotation marks. view

The optional argument view specifies where to place the banner. If view is not specified, this extension uses the active view. This extension returns the view in which the title was set.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–title
; Define a new view.
(define view (view:dl))
;; view
; Set the title of a view.
(view:set–title "Working View" view)
;; #[view 1075533160]
; "Working View" appears as the title of the view.
```

view:set–up

Scheme Extension: Viewing

Action: Sets the up vector for a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:set–up** up–direction=(0,0,1) [view=active])

Arg Types:

up–direction	gvector
view	view

Returns: gvector

Errors: None

Description: Sets the up vector for a view.

The argument up–direction specifies the up gvector of the view. view

The optional argument view specifies the view to set, and defaults to the active view. The previous up gvector is returned. The gvectors returned from the operations view:out, view:right, and view:up form a right–handed orthonormal triple. Each is the cross–product of the other two. The view:out vector is always perpendicular to the screen. After the model has been reoriented, this command determines the *y*–axis relationship to the model. The up vector should not be parallel to the vector from the target to the eye position.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:
```
; view:set–up
; Set a view's up vector.
(define view1 (view:dl))
;; view1
(view:set–up (gvector 0 0 1) view1)
;; #[gvector 0 1 0]
(view:refresh view1)
;; #[view 1075900736]
```

view:set–viewport

Scheme Extension: Viewing

Action: Sets the viewport of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax:
```
(view:set–viewport [orgx orgy width height]
    [view=active])
```

Arg Types:		
	orgx	integer
	orgy	integer
	width	integer
	height	integer
	view	view

Returns: view

Errors: None

Description: The optional argument orgx specifies the *x*–origin that sets the viewport. The optional argument orgy specifies the *y*–origin that sets the viewport. These two values locate the upper left–hand corner of the viewport.

The optional argument width specifies the horizontal length of the viewport. height

The optional argument height specifies the vertical length of the viewport. If view is not specified, the viewport of the active view is set. If no optional arguments are specified, the viewport is set to the size of the specified view, or to the size of the active view if no view is specified.

If orgx, orgy, or both are negative, the top left corner of the model will be outside the window. The scale changes depending on the relationship of width to height. The display is adjusted so the new target point is in the center of the window. view:refresh is needed to see changes to model.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set–viewport
; Define a new view.
(define view1 (view:dl))
;; view1
; Set the viewport for a view.
(view:set–viewport)
;; #[view 1075533160]
(view:set–viewport 0 0 100 100)
;; #[view 1075533160]

; Another example where the size of the view
; and viewport are changed before an output image
; is made.
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 20 20 20)))
;; block1
; Define a new view.
(define view2 (view:dl 100 200))
;; view2
; This queries the current view size. Then
; it resizes the viewport. Then it makes the
; object appear smaller by changing the width
; and height parameter of the view.
(let ((w (view:width view2)))
    (view:set–viewport 0 0 300 300 view2)
    (view:set–size (* w 2) (* w 2) view2))
;; #[view 109820984]
; Refresh the view to see the changes
(view:refresh view2)
;; #[view 109829897]
```

view:set–yon

Scheme Extension: Viewing

Action: Sets the yon clip distance of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:set-yon** yon [view=active])

Arg Types:

yon	real
view	view

Returns: view

Errors: None

Description: Specifies the yon clipping distance, which is the far distance and must be greater than the hither distance. Any item farther away from the eye than the yon distance is clipped. Hither specifies the near distance and must be greater than 0. Any item closer to the eye than the hither distance is clipped.

The optional argument view specifies the view to set. If view is not specified, the active view is used. A view's hither plane is a plane perpendicular to the view's line of sight. Objects located on the same side of the hither plane as the eye position are invisible; they are not pictured in the view. The distance along the line of sight from the eye position to its hither plane is the hither clipping distance. The default hither clipping distance is 0.025.

Hither and yon settings are very useful for perspective views or for views where the eye point is inside of the object. Establish clipping such that it clips out everything behind the eye point. It can also be used to clip out foreground and background objects.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:set-yon
; Get the yon clip distance.
(view:yon)
;; 10000
(define block1
    (solid:block (position -30 -30 -30)
    (position 20 50 20)))
;; block1
; Set the yon clipping distance.
(view:set-yon 250)
;; #[view 1075519376]
(view:refresh)
;; #[view 1075519376]
```

view:target

Scheme Extension: Viewing

Action: Gets the target position of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:target [view=active])`

Arg Types:

view	view

Returns: position

Errors: None

Description: Gets the target position of a view. The argument target–position specifies the target location of the view.

The optional argument view specifies the view to set, and defaults to the active view.

The target position needs to be different from the eye position. The resulting vector from the target to the eye position cannot be parallel to the up gvector.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:target
; Get the target position of a view.
(define view (view:dl))
;; view
(view:target view)
;; #[position 0 0 0]
```

view:up

Scheme Extension: Viewing

Action: Gets the up vector of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: `(view:up [view=active])`

Arg Types: view view

Returns: gvector

Errors: None

Description: This extension returns the unitized gvector that points toward the top side of the view. This corresponds to the *y*–axis of the view coordinate system. The gvectors returned from the operations view:out, view:right, and view:up form a right–handed orthonormal triple. Each is the cross–product of the other two. The view:out vector is always perpendicular to the screen. After the model has been reoriented, this command determines the *y*–axis relationship to the model.

The optional argument view specifies the view to set, and defaults to the active view.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:
```
; view:up
; Get the up vector for a view.
(define view (view:dl))
;; view
(view:up view)
;; #[gvector 0 1 0]
```

view:viewport

Scheme Extension: Viewing

Action: Gets the viewport origin, width, and height of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:viewport** [view=active])

Arg Types: view view

Returns: (integer integer integer integer)

Errors: None

Description: This extension returns a list of four integers: origin–x, origin–y, width, and height. The first two specify the location of the upper left–hand corner of the viewport relative to the top left corner of the associated window. The last two specify the size in pixels (width and height) from that initial point.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:
```
; view:viewport
; Get the viewport for a view.
(define view1 (view:dl))
;; view1
(view:viewport view1)
;; (0 0 600 600)
```

view:width

Scheme Extension: Viewing

Action: Gets the width of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:width** [view=active])

Arg Types:	view	view

Returns: real

Errors: None

Description: Specifies the width in pixels of the viewport from the viewports starting corner.

The optional argument view specifies the view to set, and defaults to the active view.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:
```
; view:width
; Get the width of a view.
(define view1 (view:dl))
;; view1
(view:width view1)
;; 100
```

view:with–handle

Scheme Extension: Viewing

Action: Gets the view of the specified handle.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:with-handle** handle)

Arg Types: handle integer

Returns: view | boolean

Errors: None

Description: This extension returns the view with the specified handle. If handle does not exist, this extension returns #f. Unless elements are defined, the handle is the only way to access them for later manipulation. If the handle is positive, it refers to an associated display window. If the handle is negative, it refers to a file pointer.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:with-handle
; create several windows and spread them out to make
; them all visible on screen.
(view:gl)
;; #[view 160891358]
(view:gl)
;; #[view 11076370]
; You now have 2 dl and 2 gl views displayed.
; find the active view.
(env:active-view)
;; #[view 16122558]
; delete the first gl view created.
(view:delete (view:with-handle 160891358))
;; ()
```

view:yon

Scheme Extension: Viewing

Action: Gets the yon clip distance of a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view:yon** [view=active])

Arg Types: view view

Returns: real

Errors: None

Description: Returns the yon clipping distance. Yon specifies the far distance, which must be greater than the hither distance. Any item farther away from the eye than the yon distance is clipped. Hither specifies the near distance and must be greater than 0. Any item closer to the eye than the hither distance is clipped.

The optional argument view specifies the view to set. If view is not specified, the active view is used. A view's hither plane is a plane perpendicular to the view's line of sight. Objects located on the same side of the hither plane as the eye position are invisible; they are not pictured in the view. The distance along the line of sight from the eye position to its hither plane is the hither clipping distance. The default hither clipping distance is 0.025.

Hither and yon settings are very useful for perspective views or for views where the eye point is inside of the object. Establish clipping such that it clips out everything behind the eye point. It can also be used to clip out foreground and backgrounds objects.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view:yon
; Get the yon clipping plane for a view.
(define view1 (view:dl))
;; view1
(view:yon)
;; 10000
```

view?

Scheme Extension: Viewing

Action: Determines if a Scheme object is a view.

Filename: scm/scmext/view_scm.cxx

APIs: None

Syntax: (**view?** object)

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: Refer to Action.

object is an input Scheme object.

Limitations: This extension is available on all platforms, but produces results only on NT using OpenGL.

Example:

```
; view?
; Create a new view.
(define view1 (view:dl))
;; view1
; Determine if the new view is actually a view.
(view? view1)
;; #t
(view? (position 10 20 30))
;; #f
```

void?

Scheme Extension: Mathematics

Action: Determines if a Scheme object is void.

Filename: scm/scheme/scm_typ.cxx

APIs: None

Syntax: (**void?** object)

Arg Types: object scheme–object

Returns: boolean

Errors: None

Description: Refer to Action.

object is an input Scheme object.

Limitations: None

Example:

```
; void?
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 0 0 0)))
;; block1
; Determine if the solid block is void.
(void? block1)
;; #f
```