Chapter 2.
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.

entity:count–facets

Scheme Extension: Entity, Faceting

Action: Counts the number of facets in an entity.

Filename: fct/fct_scm/fct_scm.cxx

APIs: None

Syntax: `(entity:count-facets [entity])`

Arg Types: entity entity

Returns: integer

Errors: None

Description: This extension counts the number of facets in the input entity. This procedure does not facet the entities. The user must call entity:facet to facet them or nothing prints.

Limitations: None

Example:

```
; entity:count–facets
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Facet the block.
(entity:facet block1)
;; 12
; Count the number of facets in the block.
(entity:count–facets block1)
;; 12
```

entity:delete–facets

Scheme Extension: Entity, Faceting

Action: Deletes the facets from an entity or list of entities.

Filename: fct/fct_scm/fct_scm.cxx

APIs: None

Syntax: (**entity:delete–facets** entity–list)

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

Returns: (entity ...)

Errors: None

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

Limitations: None

Example:

```
; entity:delete–facets
; Create and facet a block, then delete the facets.
(define block1
    (solid:block (position 0 0 0)
    (position 20 20 20)))
;; block1
(entity:facet block1)
;; 12
(entity:delete–facets block1)
;; #[entity 2 1]
(entity:count–facets block1)
;; 12
```

entity:display–facets

Scheme Extension: Entity, Faceting

Action: Displays the facets of an entity or list of entities.

Filename: fct/dfct_scm/dfct_scm.cxx

APIs: None

Syntax: (**entity:display–facets** entity–list [view] [hide])

Arg Types:

entity–list	entity | (entity ...)
view	view
hide	boolean

Returns: integer

Errors: None

Description: This extension returns the number of facets displayed from the input entity–list.

The optional argument view specifies the view to display the entity; the default is the active view.

If the optional hide argument is #t, the back facing polygons are not displayed. A polygon is considered back facing if the surface normal at all vertices points away from the viewer.

Limitations: None

Example:

```
; entity:display–facets
; Create and facet a block, then display the facets.
(define block1
    (solid:block (position 0 0 0)
    (position 25 25 25)))
;; block1
; Facet the solid block.
(entity:facet block1)
;; 12
; Display the facets of the solid block.
(entity:display–facets block1)
;; ()
```

entity:facet

Scheme Extension: Entity, Faceting

Action: Facets an entity or list of entities.

Filename: fct/fct_scm/fct_scm.cxx

APIs: api_get_facet_edge_points

Syntax:
```
(entity:facet entity–list [view] [unfaceted–only]
    [acis–opts])
```

Arg Types:

entity–list	entity | (entity ...)
view	view
unfaceted–only	boolean
acis–opts	acis–options

Returns: integer

Errors: None

Description: The argument entity–list specifies the facets of an entity or list of entities to save. The facets produced become part of the part. When facets are saved with the part, refinements cannot be modified unless the entity:delete–facets extension is applied. New refinement modifications must then be faceted with the entity to be saved with that entity.

The optional argument view specifies which view to facet the entities, and it is usually used when view–dependent refinements have been set; the default is the active view.

When the optional argument unfaceted–only is set to #t, only unfaceted geometry is faceted. This extension returns the number of facets.

The optional argument acis–opts can be used to enable journaling and versioning options.

Limitations: None

Example:
```
; entity:facet
; Create a block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 18 10)))
;; block1
; Facet the block.
(entity:facet block1)
;; 12
(entity:display–facets block1)
;; ()
```

entity:facet–area

Scheme Extension: Entity, Faceting

Action: Returns the area of the facets currently attached to the entity.

Filename: fct/fct_scm/fct_scm.cxx

APIs: api_facet_area

Syntax: `(entity:facet–area entity–list [acis–opts])`

Arg Types:

entity–list	entity | (entity ...)
acis–opts	acis–options

Returns: real

Errors: None

Description: Gets the area of the facets currently attached to the entity. The entity must be already faceted for this to be effective. Any parts of the entity not faceted will be ignored.

The argument entity–list specifies the facets of an entity or list of entities.

The optional argument acis–opts can be used to enable journaling and versioning options.

Limitations: None

Example:

```
; entity:facet–area
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 9))
;; sphere1
; Facet the sphere.
(entity:facet sphere1)
;; 698
; Calculate the area of the facets of the sphere.
(entity:facet–area sphere1)
;; 1010.65030563994
```

entity:faceted?

Scheme Extension: Entity, Faceting

Action: Determines if an entity is faceted.

Filename: fct/fct_scm/fct_scm.cxx

APIs: None

Syntax: `(entity:faceted? entity)`

Arg Types: entity entity

Returns: boolean

Errors: None

Description: This extension returns #t if the entity is a face and has been faceted or if the entity is a higher–level entity and all the faces it contains have been faceted; otherwise, it returns #f.

Limitations: None

Example:

```
; entity:faceted?
; Create a cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 8 8 8) 20))
;; cyl1
; Determine if the cylinder is faceted.
(entity:faceted? cyl1)
;; #f
; Facet the cylinder.
(entity:facet cyl1)
;; 96
(entity:display–facets cyl1)
;; ()
; Determine if the cylinder is faceted.
(entity:faceted? cyl1)
;; #t
```

entity:print–facets

Scheme Extension: Entity, Faceting

Action: Prints the facets of an entity or list of entities.

Filename: fct/dfct_scm/dfct_scm.cxx

APIs: None

Syntax:

```
(entity:print–facets [entity–list]
    [view] [scale] [hide] [acis–opts])
```

Arg Types:	entity–list	entity | (entity ...)
	view	view
	scale	real
	hide	boolean
	acis–opts	acis–options

Returns: unspecified

Errors: None

Description: This extension prints the facets of an entity or a list of entities. If no entities are specified, then all entities that are both displayed and faceted are printed. This procedure does not facet the entities. The user must call entity:facet to facet them or nothing prints.

The optional argument view is used as the base view to determine the orientation of the printed display. It defaults to the active view.

The optional argument scale controls the size of the image. If scale is not specified, then the area printed is the same as the size of the base view. For example, if the base view is 300 x 300 pixels, an area of 300x300 dots prints. If the printer resolution is 300 dots per inch, this means that the size of the printed image is 1 square inch. If scale is 5 in this example, then the size of the image is 5 inches by 5 inches.

If the optional hide argument is #t, the back facing polygons are not printed. A polygon is considered back facing if the surface normal at all vertices points away from the viewer.

The optional argument acis–opts can be used to enable journaling and versioning options.

Limitations: Only available on Windows and Windows NT platforms.

Example:

```
; entity:print–facets
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Facet the block.
(entity:facet block1)
;; 12
(entity:display–facets block1)
;; ()
; Print the list of facets.
(entity:print–facets block1)
;; ()
```

entity:refinement

Scheme Extension: Entity, Faceting

Action: Gets an entity's faceting refinement.

Filename: fct/fct_scm/fct_scm.cxx

APIs: api_get_entity_refinement

Syntax: `(entity:refinement entity [acis–opts])`

Arg Types:

entity	entity
acis–opts	acis–options

Returns: refinement | boolean

Errors: None

Description: This extension returns the entity's refinement. If no refinement is assigned, this extension returns #f.

The types of valid refinements to control the faceting are (refer to class REFINEMENT for default values):

“surface tolerance”	is the maximum distance between facet and true surface.
“last used surface tolerance”	specifies the previously set surface tolerance.
“normal tolerance”	is the maximum difference between any two normals of facet.
“aspect ratio”	is the approximate aspect ratio of each cell in grid.
“max edge length”	is the maximum size of an edge of a facet.
“min v grid lines”	is the minimum number v grid lines used to display a face.
“min u grid lines”	is the minimum number u grid lines used to display a face.
“max grid lines”	limits the grid lines of a parametric grid for the surface of a face.
“grading”	adjusts facets to get better parametric aspect ratio of a grid cell.
“grid mode”	specifies whether a grid is used and whether the points where the grid cuts the edge is inserted to the edge.
“triang mode”	specifies how much triangulation to do.
“adjust mode”	specifies type of triangle smoothing to do. Determines if facet nodes should be adjusted for smoothing. Also determines if the grid points should be adjusted or not.
“surf mode”	specifies the type of surface.

Available surface types are:

AF_SURF_ALL	all surface type
AF_SURF_REGULAR	a surface with planar cells
AF_SURF_IRREGULAR	a surface with possibly nonplanar cells
AF_SURF_PLANE	a planar surface
AF_SURF_CONE	a cylindrical or conical surface
AF_SURF_SPHERE	a spherical surface
AF_SURF_TORUS	toroidal surface
AF_SURF_SPLINE	a spline surface

The optional argument acis–opts can be used to enable journaling and versioning options.

Limitations: None

Example:

```
; entity:refinement
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 30 30 0) 20))
;; cyl1
; Create a refinement.
(define refine1 (refinement))
;; refine1
; Get the default refinement values.
(refinement:props refine1)
;; (("surface tolerance" . –1)
;; ("normal tolerance" . 15)
;; ("aspect ratio" . 0)
;; ("max edge length" . 0)
;; ("min v_grid lines" . 0)
;; ("min u_grid lines" . 0)
;; ("max grid lines" . 3000)
;; ("grading" . #f)
;; ("grid mode" . "AF_GRID_INTERIOR")
;; ("triang mode" . "AF_TRIANG_FRINGE_2")
;; ("adjust mode" . "AF_ADJUST_NONE")
;; ("surf mode" . "AF_SURF_ALL"))
; Set the properties of the refinement.
(refinement:set–prop refine1 "surface tolerance" 0.5)
;; ()
; Attach the refinement to the cylinder.
(define refine (entity:set–refinement cyl1 refine1))
;; refine
; Get the cylinder's refinement.
(define get–refine (entity:refinement cyl1))
;; get–refine
```

entity:refinements

Scheme Extension: Entity, Faceting

Action: Gets all the faceting refinements attached to an ENTITY.

Filename: fct/fct_scm/fct_scm.cxx

APIs:	None
Syntax:	`(`**`entity:refinements`** `entity)`
Arg Types:	entity entity
Returns:	(refinement ...) | boolean
Errors:	None
Description:	This extension returns a list of all the refinements attached to an ENTITY. Refer to Scheme extension entity:refinement for more information.
Limitations:	None

Example:

```
; entity:refinements
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 30 30 0) 20))
;; cyl1
; Create a refinement.
(define refine1 (refinement))
;; refine1
; Get the default refinement values.
(refinement:props refine1)
;; ((”surface tolerance” . –1)
;; (”normal tolerance” . 15)
;; (”aspect ratio” . 0) (”max edge length” . 0)
;; (”min v_grid lines” . 0)
;; (”min u_grid lines” . 0)
;; (”max grid lines” . 3000)
;; (”grading” . #f)
;; (”grid mode” . ”AF_GRID_INTERIOR”)
;; (”triang mode” . ”AF_TRIANG_FRINGE_2”)
;; (”adjust mode” . ”AF_ADJUST_NONE”)
;; (”surf mode” . “AF_SURF_ALL”))
; Set the properties of the refinement.
(refinement:set–prop refine1 ”normal tolerance” 15)
;; ()
; Attach the refinement to the cylinder.
(define attach (entity:set–refinement cyl1 refine1))
;; attach
; Get the cylinder’s refinement.
(entity:refinement cyl1)
;; #[entity 3 1]
; Get all the cylinder’s refinements.
(define all (entity:refinements cyl1))
;; all
```

entity:set–facet–color

Scheme Extension: Entity, Faceting, Colors

Action: Sets the color information to each facet vertex based on the input vector.

Filename: fct/fct_scm/vtmp_scm.cxx

APIs: None

Syntax: (**entity:set–facet–color** entity vector)

Arg Types:	entity vector	entity gvector
Returns:	entity	
Errors:	None	

Description: This extension sets the color information to each facet vertex based on the input vector. The newly created VERTEX_TEMPLATE class sets up storage for the tokens used for position, normal, color, and *uv*. When the model is subsequently faceted, the polygon's vertices contain data for each of these tokens.

Limitations: Visible only with Advanced Rendering Component linked in.
The model must have a material with a "base" color shader type.

Example:

```
; entity:set–facet–color
; Set up the default vertex template
(vertex–template:set–default)
;   Use–count   :   0
;   Tokens      :
;       POSITION_TOKEN
;       NORMAL_TOKEN
;       COLOR_TOKEN
;       UV_TOKENS
;; #t
(define block1
    (solid:block (position –10 –10 –10)
    (position 5 10 15)))
;; block1
; Set up the material for use.
(define mat1 (material))
;; mat1
(material:set–color–type mat1 ”base”)
;; ()
(entity:set–material block1 mat1)
;; ()
; Facet the block.
(entity:facet block1)
;; 12
; Set the facet color.
(define color (entity:set–facet–color block1
    (gvector 1 1 1)))
;; color
```

entity:set–refinement

Scheme Extension: Entity, Faceting

Action: Attaches a faceting refinement to an entity or list of entities.

Filename: fct/fct_scm/fct_scm.cxx

APIs: api_set_entity_refinement

Syntax: `(entity:set–refinement entity–list refine [acis–opts])`

Arg Types:		
	entity–list	entity | (entity ...)
	refine	refinement | boolean
	acis–opts	acis–options

Returns: (entity ...)

Errors: None

Description: The argument entity–list specifies an entity or list of entities to attach a refinement.

The argument refine specifies the refinement entity to attach or #f to exclude an entity from an entity list.

The optional argument acis–opts can be used to enable journaling and versioning options.

This extension returns the input entity–list. Refer to Scheme extension entity:refinement for more information.

Limitations: None

Example:

```
; entity:set–refinement
; Create a solid cylinder.
(define cyl1
    (solid:cylinder (position 0 0 0)
    (position 8 8 0) 20))
;; cyl1
; Create a refinement.
(define refine1 (refinement))
;; refine1
; Get the default refinement values.
(refinement:props refine1)
;; ((”surface tolerance” . –1)
;; (”normal tolerance” . 15)
;; (”aspect ratio” . 0) (”max edge length” . 0)
;; (”min v_grid lines” . 0)
;; (”min u_grid lines” . 0)
;; (”max grid lines” . 3000) (”grading” . #f)
;; (”grid mode” . ”AF_GRID_INTERIOR”)
;; (”triang mode” . ”AF_TRIANG_FRINGE_2”)
;; (”adjust mode” . ”AF_ADJUST_NONE”)
;; (”surf mode” . “AF_SURF_ALL”))
; Set the properties of the refinement.
(refinement:set–prop
    refine1 ”aspect ratio” 0.5)
;; ()
; Set the refinement on the cylinder.
(define refine (entity:set–refinement cyl1 refine1))
;; refine
; Remove the refinement from the cylinder.
(define remove (entity:set–refinement cyl1 #f))
;; remove
```

entity:write–facets

Scheme Extension: Entity, Faceting

Action: Writes facets to a file.

Filename: fct/fct_scm/fct_scm.cxx

APIs: None

Syntax: (**entity:write–facets** entity–list filename)

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

Returns: boolean

Errors: None

Description: The argument entity–list specifies the entity or list of entities to include facet information.

The argument filename specifies the name of the file to create.

This extension returns #t if the operation is successful.

Limitations: None

Example:

```
; entity:write–facets
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 15 15 15)))
;; block1
; Facet the block.
(entity:facet block1)
;; 12
; Write the list of facets to a file.
(entity:write–facets block1 ”eraseme”)
;; ()
```

entity:write–raw–facets

Scheme Extension: Entity, Faceting

Action: Writes facets to a file.

Filename: fct/fct_scm/fct_scm.cxx

APIs: None

Syntax: (**entity:write–raw–facets** entity–list filename)

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

Returns: boolean

Errors: None

Description: The argument entity–list specifies the entity or list of entities to include facet information.

The argument filename specifies the name of the file to create.

This extension returns #t if the operation is successful. The format of the file is one triangle per line as follows: v1.X v1.Y v1.Z v2.X v2.Y v2.Z v3.X v3.Y v3.Z

Limitations: None

Example:

```
; entity:write-raw-facets
; Create a solid block.
(define block1(solid:block (position 0 0 0)
    (position 15 15 15)))
;; block1
; Facet the block.
(entity:facet block1)
;; 12
; Write the list of facets to a file.
(entity:write-raw-facets block1 "eraseme")
;; ()
```

refinement

Scheme Extension: Faceting

Action: Creates a refinement for the active part.

Filename: fct/fct_scm/fct_scm.cxx

APIs: None

Syntax: (**refinement**)

Arg Types: None

Returns: refinement

Errors: None

Description: A refinement is an entity created in the active part that controls how closely the surfaces of entities are approximated by their faceted representation. A refinement is associated with a face, shell, lump, or solid body. The properties of the refinement are set using refinement:set–prop. Refer to Scheme extension entity:refinement for more information.

Limitations: None

Example:

```
; refinement
; Create a new refinement.
(define refine (refinement))
;; refine
```

refinement:default

Scheme Extension: Faceting

Action: Gets the default refinement.

Filename: fct/fct_scm/fct_scm.cxx

APIs: api_get_default_refinement

Syntax: (**refinement:default**)

Arg Types: None

Returns: refinement

Errors: None

Description: This extension returns the current default refinement. Refer to Scheme extension entity:refinement for more information. Refer to class REFINEMENT for the default values.

Limitations: None

Example:

```
; refinement:default
; Create a new refinement.
(define refine1 (refinement))
;; refine1
; Set the properties of refine.
(refinement:set–prop refine1 ”max edge length” 11)
;; ()
; Determine the properties of refine.
(refinement:props refine1)
;; ((”surface tolerance” . –1)
;; (”normal tolerance” . 15)
;; (”aspect ratio” . 0) (”max edge length” . 11)
;; (”min v_grid lines” . 0)
;; (”min u_grid lines” . 0)
;; (”max grid lines” . 3000) (”grading” . #f)
;; (”grid mode” . ”AF_GRID_INTERIOR”)
;; (”triang mode” . ”AF_TRIANG_FRINGE_2”)
;; (”adjust mode” . ”AF_ADJUST_NONE”)
;; (”surf mode” . “AF_SURF_ALL”))
; Set refine as the default.
(refinement:set–default refine1)
;; ()
; Create a new refinement with default properties.
(define refine2 (refinement))
;; refine2
; Get the default refinement
(define refine (refinement:default))
;; refine
```

refinement:props

Scheme Extension: Faceting

Action: Gets the current property list of a refinement.

Filename: fct/fct_scm/fct_scm.cxx

APIs: None

Syntax: (**refinement:props** refinement)

Arg Types: refinement refinement

Returns: ((string . real | integer) ...)

Errors: None

Description: This extension returns a list of pairs of refinement criteria and their values.

Limitations: None

Example:
```
; refinement:props
; Create a refinement.
(define refine1 (refinement))
;; refine1
; Determine the properties of a refinement.
(refinement:props refine1)
;; ((”surface tolerance” . –1)
;; (”normal tolerance” . 15)
;; (”aspect ratio” . 0) (”max edge length” . 0)
;; (”min v_grid lines” . 0)
;; (”min u_grid lines” . 0)
;; (”max grid lines” . 3000) (”grading” . #f)
;; (”grid mode” . ”AF_GRID_INTERIOR”)
;; (”triang mode” . ”AF_TRIANG_FRINGE_2”)
;; (”adjust mode” . ”AF_ADJUST_NONE”)
;; (”surf mode” . “AF_SURF_ALL”))
```

refinement:set–default

Scheme Extension: Faceting

Action: Sets the default refinement to the specified refinement.

Filename: fct/fct_scm/fct_scm.cxx

APIs: api_set_default_refinement

Syntax: (**refinement:set–default** refinement [acis–opts])

Arg Types:		
	refinement	refinement
	acis–opts	acis–options

Returns: unspecified

Errors: None

Description: Input argument refinement is set as the default refinement.

The optional argument acis–opts can be used to enable journaling and versioning options.

Limitations: None

Example:

```
; refinement:set-default
; Create a refinement.
(define refine1 (refinement))
;; refine1
; Set the default refinement.
(refinement:set-default refine1)
;; ()
```

refinement:set–prop

Scheme Extension: Faceting, Modeler Control

Action: Sets a property of a refinement.

Filename: fct/fct_scm/fct_scm.cxx

APIs: None

Syntax:

```
(refinement:set-prop refinement [name value |
    name-value-list])
```

Arg Types:

name	string
refinement	refinement
value	real | integer
name–value–list	pair

Returns: real | integer

Errors: None

Description: refinement specifies a refinement entity.

name specifies the name of property to set.

value specifies the new value for the named property.

This extension returns the property's previous value. Refer to Scheme extension entity:refinement for more information.

Limitations: None

Example:

```
; refinement:set–prop
; Create a new refinement.
(define refine1 (refinement))
;; refine1
; Set a property of the refinement.
(refinement:set–prop
    refine1 ”aspect ratio” 0.5)
;; ()
; Determine the properties of a refinement.
(refinement:props refine1)
;; ((”surface tolerance” . –1)
;; (”normal tolerance” . 15)
;; (”aspect ratio” . 0.5) (”max edge length” . 0)
;; (”min v_grid lines” . 0)
;; (”min u_grid lines” . 0)
;; (”max grid lines” . 3000) (”grading” . #f)
;; (”grid mode” . ”AF_GRID_INTERIOR”)
;; (”triang mode” . ”AF_TRIANG_FRINGE_2”)
;; (”adjust mode” . ”AF_ADJUST_NONE”)
;; (”surf mode” . “AF_SURF_ALL”))
; Set the default property of the refinement.
(refinement:set–default refine1)
;; ()
```

refinement?

Scheme Extension: Faceting

Action: Determines if a Scheme object is a refinement.

Filename: fct/fct_scm/fct_scm.cxx

APIs: None

Syntax: (**refinement?** object)

Arg Types:	object	scheme–object

Returns: boolean

Errors: None

Description: Refer to Action.

Limitations: None

Example:

```
; refinement?
; Create a refinement.
(define refine1 (refinement))
;; refine1
; Determine the refinement is actually a refinement.
(refinement? refine1)
;; #t
```

vertex–template:set–default

Scheme Extension: Faceting

Action: Creates a default vertex template for use with faceting.

Filename: fct/fct_scm/vtmp_scm.cxx

APIs: api_set_default_vertex_template

Syntax: (**vertex–template:set–default**)

Arg Types: None

Returns: boolean

Errors: None

Description: This extension creates a data structure, VERTEX_TEMPLATE. The newly created structure sets up storage for the tokens used for position, normal, color, and *uv*. When the model is subsequently faceted, the polygon's vertices contain data for each of these tokens.

Limitations: Visible only with Advanced Rendering Component linked in. The model must have a material with a "base" color shader type.

Example:

```
; vertex–template:set–default
; Set up the default vertex template.
(vertex–template:set–default)
;   Use–count   :   0
;   Tokens      :
;       POSITION_TOKEN
;       NORMAL_TOKEN
;       COLOR_TOKEN
;       UV_TOKENS
;; #t
; Create a solid block.
(define block1
    (solid:block (position –10 –10 –10)
    (position 5 10 15)))
;; block1
; Set up a material.
(define mat1 (material))
;; mat1
(material:set–color–type mat1 ”blue marble”)
;; ()
(entity:set–material block1 mat1)
;; ()
; Facet the block.
(entity:facet block1)
;; 12
; Set the facet color.
(define color (entity:set–facet–color block1
    (gvector 1 1 1)))
;; color
```