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.

cell:2d?

Scheme Extension: Cellular Topology

Action: Determines if the given cell is of the type CELL2D.

Filename: ct/ct_scm/cell_scm.cxx

APIs: None

Syntax: `(cell:2d? cell)`

Arg Types: cell — cell

Returns: boolean

Errors: None

Description: This extension returns #t if the input cell is of the type CELL2D; otherwise, it returns #f.

A CELL2D is a topology entity. It is associated with a lump and consists of a set of faces: DOUBLE_SIDED and BOTH_OUTSIDE. A CELL2D is created using cell:attach with respect to a sheet or mixed-dimension body.

Limitations: None

Example:

```
; cell:2d?
; Create a planar face.
(define face1 (face:plane (position 0 0 0) 10 10))
;; face1
; Make the planar face a 2D cell.
(define sheet1 (sheet:2d (sheet:face face1)))
;; sheet1
; Attach cellular topology to each lump.
(define cell1 (cell:attach sheet1))
;; cell1
; Determine if the cell is actually a cell.
(cell? (car cell1))
;; #t
; Determine if the cell is a 2D cell.
(cell:2d? (car cell1))
;; #t
```

cell:3d?

Scheme Extension: Cellular Topology

Action: Determines if the input cell is of the type CELL3D.

Filename: ct/ct_scm/cell_scm.cxx

APIs: None

Syntax: (**cell:3d?** cell)

Arg Types:	cell	cell

Returns: boolean

Errors: None

Description: This extension returns #t if the given cell is of the type CELL3D; otherwise, it returns #f.

A CELL3D is a topology entity. It is a subportion of a lump and is bounded by faces. The faces are either SINGLE_SIDED, or DOUBLE_SIDED with BOTH_INSIDE containment. A CELL3D is created using cell:attach with respect to a solid or mixed-dimension body.

Limitations: None

Example:

```
; cell:3d?
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Attach cellular topology to each lump.
(define cell1 (cell:attach block1))
;; cell1
; Determine if the cellular topology is a cell.
(cell? (car cell1))
;; #t
; Determine if the cellular topology is 3D cell.
(cell:3d? (car cell1))
;; #t
```

cell:area

Scheme Extension: Cellular Topology

Action: Gets the total surface area of all faces in the input cell.

Filename: ct/ct_scm/cell_scm.cxx

APIs: api_ct_cell_area

Syntax: (**cell:area** cell [accuracy=0.1])

Arg Types:

cell	cell
accuracy	real

Returns: pair (real . real)

Errors: None

Description: This extension finds the total surface area for all faces in the input cell. If cell is of the type CELL3D, this is the area of all faces used once in the cell. If cell is of the type CELL2D, this is the area of both sides of each face.

The optional accuracy parameter is a percentage used in the area calculation. If it is not specified, it defaults to 0.1%.

This extension returns the pair: area and accuracy–achieved.

The accuracy achieved is always 0 if all faces in the cell are analytic surfaces and the area has been determined to be within the limits of accuracy of the computer. Several faces have their areas evaluated analytically. These include:

– A Planar face bounded by linear and/or elliptical edges.
– A conical face bounded by linear and/or circular edges and part of the curved surface of a circular cross-section.
– A cylindrical face bounded by linear and/or elliptical edges and part of the curved surface of a circular cross-section.
– A spherical face bounded by circular edges in the longitudinal or latitudinal planes and either a full sphere surface or a part of such surface.

The accuracy achieved is an estimate of the maximum relative error in the area calculation based on the maximum difference between the calculated value and the true value as a fraction of the true value. In the case of a CELL2D, the areas of both sides of all its faces is used in the calculation. In the case of a CELL3D, the areas of all faces in the CELL3D are used just once in the calculation.

Limitations: None

Example:

```
; cell:area
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Attach the solid block to a cell.
(define cell1 (cell:attach block1))
;; cell1
; Find the area of the cell.
(cell:area (car cell1))
;; (9600 . 0)
; Find the area of the cell
; with the specified accuracy.
(cell:area (car cell1) 0.5)
;; (9600 . 0)
```

cell:attach

Scheme Extension: Cellular Topology

Action: Attaches cellular topology to each lump within each body in the input list.

Filename: ct/ct_scm/cell_scm.cxx

APIs: api_ct_attach

Syntax: (**cell:attach** entity–list)

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

Returns: entity ...

Errors: None

Description: This extension attaches cellular topology to each lump within each solid or sheet body entity in the given input entity–list. The created cells may be a sheet body (CELL2D), a solid body (CELL3D), or a combination of both. Returns a list of 2D or 3D cell entities.

Limitations: None

Example:

```
; cell:attach
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Attach the solid block to a cell.
(define cell1 (cell:attach block1))
;; cell1
```

cell:classify–position

Scheme Extension: Cellular Topology

Action: Determines the relationship of a given point to a given cell.

Filename: ct/ct_scm/cell_scm.cxx

APIs: api_ct_point_in_cell

Syntax: (**cell:classify–position** cell position)

Arg Types:	cell position	cell position

Returns: integer

Errors: None

Description: This extension determines if a given point (position) is inside, outside, or on the specified cell. cell must be of the type CELL3D.

This extension returns

–1, if the point is inside the cell.
0, if the point is on the cell.
1, if the point is outside the cell.

Limitations: None

Example:
```
; cell:classify-position
; Create a solid block.
(define block1
    (solid:block (position -20 -20 -20)
    (position 20 20 20)))
;; block1
; Attach the block to a cell.
(define cell1 (cell:attach block1))
;; cell1
; Determine if the following positions
; are on the cell.
(cell:classify-position (car cell1)
    (position 0 0 0))
;; -1
(cell:classify-position (car cell1)
    (position 20 0 0))
;; 0
(cell:classify-position (car cell1)
    (position 40 0 0))
;; 1
```

cell:copy

Scheme Extension: Cellular Topology

Action: Creates a new body that is a copy of the given cell.

Filename: ct/ct_scm/cell_scm.cxx

APIs: api_ct_copy_cell

Syntax: (**cell:copy** cell)

Arg Types:	cell	cell

Returns: pair

Errors: None

Description: This extension makes a copy of the given cell by creating a one–lump body. The copy is a solid body for a CELL3D or a sheet body for a CELL2D. The newly-created body has cellular topology entity attached to its lump.

This extension returns the pair (new–body . new–cell).

Limitations: None

Example:
```
; cell:copy
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Attach the solid block to a cell.
(define cell1 (cell:attach block1))
;; cell1
; Copy the cell.
(define copy (cell:copy (car cell1)))
;; copy
```

cell:expand

Scheme Extension: Cellular Topology

Action: Expands the cellular topology by grouping cells into supercells.

Filename: ct/ct_scm/cell_scm.cxx

APIs: api_ct_expand

Syntax: (**cell:expand** entity–list)

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

Returns: entity ...

Errors: None

Description: This extension expands the cells in the sheet or solid bodies in the entity–list into supercells. This organizes the cells into a hierarchy based on spatial proximity, which allows faster cell-based operations. This extension does not try to expand the cell list into supercells unless 50 or more cells exist. The return list is the input list.

Limitations: None

Example:

```
; cell:expand
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 30))
;; sphere1
; Attach the sphere to a cell.
(define cell1 (cell:attach sphere1))
;; cell1
; Expand the cells into supercells.
(define expand (cell:expand sphere1))
;; expand
(part:clear)
;; #t

; cell:expand
; Additional example,
; Create a lattice with more cells.
(define lattice1 (letrec
    ((loop (lambda (one two three acc)
        (if (= one 3)
            (apply bool:nonreg-unite acc)
            (if (= two 3)
                (loop (+ one 1) 0 0 acc)
                (if (= three 3)
                    (loop one (+ two 1) 0 acc)
                        (let ((org-pos
                            (position (* one 5)
                            (* two 5)
                            (* three 5))))
                    (loop
                        one two (+ three 1)
                        (cons (solid:block org-pos
                            (position:offset org-pos
                            (gvector 10 10 10)))
            acc))))))))) (loop 0 0 0 '())))
;; lattice1
(define cells1 (cell:attach lattice1))
;; cells1
; Expand the cells into supercells.
(define expand (cell:expand lattice1))
;; expand
```

cell:find

Scheme Extension: Cellular Topology

Action: Finds the cell associated with a given face.

Filename: ct/ct_scm/cell_scm.cxx

APIs: None

Syntax: (**cell:find** face1 [face2])

Arg Types:

face1	entity
face2	entity

Returns: cell

Errors: None

Description: The face1 argument is generally the result of a pick operation and is part of the cell in question.

The optional face2 is a face that is part of the same cell.

The algorithm determines the face's sense (front and back) for face1 and face2. This is used to create a cell relationship. The cells are tested and the proper active cell is returned. This is used as part of graph theory, selective Booleans, and partial sweeping to select the graph elements to keep.

Limitations: None

Example:

```
; cell:find
; Create a selective boolean example.
(define blank (solid:block (position 0 0 0)
    (position 25 10 10)))
;; blank
; blank => #[entity 2 1]
(define b2 (solid:block (position 5 0 0)
    (position 10 5 10)))
;; b2
(define b3 (solid:block (position 15 0 0)
    (position 20 5 10)))
;; b3
(define subtractb2 (solid:subtract blank b2))
;; subtractb2
(define subtractb3 (solid:subtract blank b3))
;; subtractb3
(define tool (solid:cylinder
    (position –5 2.5 5) (position 30 2.5 5)1))
;; tool
(define g (bool:select1 blank tool))
;; g
; Find the face entities of the blank.
(define faces1 (entity:faces blank))
;; faces1
; Pick out the fourth face to locate its cell
(define fourth–face (list–ref faces1 3))
;; fourth–face
; Locate the cell associated with this face
(cell:find fourth–face)
;; #[entity 35 1]
; Verify that this is one of the cells.
(define cells (entity:cells blank))
;; cells
; To continue the selective boolean example,
; create a list of cells to keep. This list is what
; gets passed to bool:select2.
```

cell:flatten

Scheme Extension: Cellular Topology

Action: Flattens the cellular topology by removing any supercells.

Filename: ct/ct_scm/cell_scm.cxx

APIs: api_ct_flatten

Syntax: `(cell:flatten entity–list)`

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

Returns: entity ...

Errors: None

Description: The cell:expand extension expands the cells in the sheet or solid bodies in the entity–list into supercells. The cell:flatten extension removes any supercells from the cellular topology attached to each sheet or solid body in the input entity–list. The return value is the input list.

Limitations: None

Example:

```
; cell:flatten
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 30))
;; sphere1
; Attach the sphere to a cell.
(define cell1 (cell:attach sphere1))
;; cell1
; Expand the cells into supercells.
(define expand (cell:expand sphere1))
;; expand
; Remove any supercells from the cellular topology.
(define flatten (cell:flatten sphere1))
;; flatten
```

```
; cell:flatten
; Additional example,
; Create a lattice with more cells.
(define lattice1 (letrec
    ((loop (lambda (one two three acc)
        (if (= one 3)
            (apply bool:nonreg-unite acc)
            (if (= two 3)
                (loop (+ one 1) 0 0 acc)
                (if (= three 3)
                    (loop one (+ two 1) 0 acc)
                        (let ((org-pos
                            (position (* one 5)
                            (* two 5)
                            (* three 5))))
                    (loop one two (+ three 1)
                        (cons (solid:block org-pos
                            (position:offset org-pos
                            (gvector 10 10 10)))
            acc))))))))) (loop 0 0 0 '())))
;; lattice1
(define cells1 (cell:attach lattice1))
;; cells1
; Expand the cells into supercells.
(define expand1 (cell:expand lattice1))
;; expand1
; Remove any supercells from the cellular topology.
(define flatten (cell:flatten lattice1))
;; flatten
```

cell:massprop

Scheme Extension: Cellular Topology

Action: Gets the mass properties of the input cell.

Filename: ct/ct_scm/cell_scm.cxx

APIs: api_ct_cell_mass_pr

Syntax:
```
(cell:massprop cell [selector accuracy
    proj-root proj-normal])
```

Arg Types:	cell	cell
	selector	integer
	accuracy	real
	proj–root	position
	proj–normal	vector

Returns: (string ...) . (real ...)

Errors: None

Description: This extension determines the mass properties of the input cell, which must be of the type CELL3D.

The optional selector argument is an integer: 0, 1, or 2; the default is 1. The selector values determine the type of mass property calculations performed and returned as shown below:

0 indicates volume and accuracy achieved.
1 indicates volume, accuracy achieved, and center of gravity.
2 indicates volume, accuracy achieved, center of gravity, inertia matrix, principle moments, and principle axes.

The optional accuracy argument is a percentage used in the mass property calculation. If it is not specified, the default is 0.1%.

The optional arguments proj–root for a position and proj–norm for a normal vector specify a projection plane for the calculations. The default for the proj–root argument is the center of the box that bounds the body. The default for the proj–norm normal vector is in the direction of the z–axis.

The return value is an associative list based on the value of the selector.

Limitations: None

Example:

```
; cell:massprop
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Attach the block to a cell.
(define cell1 (cell:attach block1))
;; cell1
; Get a list of the cell mass properties.
(cell:massprop (car cell1) 2)
;; ((”volume” . 64000) (”accuracy achieved” . 0)
;; (”center of mass” . #[position 0 0 0])
;; (”principal moment x” . 17066666.6666667)
;; (”principal axis x” . #[gvector 1 0 0])
;; (”principal moment y” . 17066666.6666667)
;; (”principal axis y” . #[gvector 0 1 0])
;; (”principal moment z” . 17066666.6666667)
;; (”principal axis z” . #[gvector 0 0 1])
;; (”inertia tensor” 17066666.6666667 0 0 0
;; 17066666.6666667 0 0 0 17066666.6666667))
```

cell:remove

Scheme Extension: Cellular Topology

Action: Removes cellular topology attached to any lump within each body in the input list.

Filename: ct/ct_scm/cell_scm.cxx

APIs: api_ct_remove

Syntax: (**cell:remove** entity–list)

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

Returns: entity ...

Errors: None

Description: This extension removes cellular topology attached to any LUMP within each sheet or solid body in the input entity–list. The return value is the input list.

Limitations: None

Example:

```
; cell:remove
; Create a solid block.
(define block1
    (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 30))
;; sphere1
; Attach the solid block and sphere to a cell.
(define cell1 (cell:attach
    (list block1 sphere1)))
;; cell1
; (#[entity 4 1] #[entity 5 1])
; Remove the solid block from cell1.
(define remove (cell:remove block1))
;; remove
; Verify the block has been removed.
cell1
;; (#[(deleted) entity 4] #[entity 5 1])
```

cell?

Scheme Extension: Cellular Topology

Action: Determines if the given entity is of the type cell.

Filename: ct/ct_scm/cell_scm.cxx

APIs: None

Syntax: (**cell?** entity)

Arg Types:	entity	entity

Returns: boolean

Errors: None

Description: This extension returns #t if the given entity is of the type cell; otherwise, it returns #f.

Limitations: None

Example:

```
; cell?
; Create a solid block.
(define block1 (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; block1
; Create a solid sphere.
(define sphere1 (solid:sphere (position 0 0 0) 30))
;; sphere1
; Attach the solid block and sphere to a cell.
(define cell1 (cell:attach
    (list block1 sphere1)))
;; cell1
; Determine if the solid block is a cell type.
(cell? block1)
;; #f
; Determine if the lump is a cell type.
(cell? (car cell1))
;; #t
(cell? (car (cdr cell1)))
;; #t
```

entity:cells

Scheme Extension: Cellular Topology

Action: Determines the list of cells associated an entity.

Filename: ct/ct_scm/cell_scm.cxx

APIs: None

Syntax: (**entity:cells** body–list)

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

Returns: (entity ...)

Errors: None

Description: This Scheme extension is very useful when doing selective Booleans (e.g., bool:select1 and bool:select2, which determine what should be kept and what should be thrown away by the cells that are selected. entity:cells returns a list of entities that are cells.

body–list is an input list of entities.

Limitations: None

Example:

```
; entity:cells
; Create a selective boolean example.
(define blank (solid:block (position 0 0 0)
    (position 25 10 10)))
;; blank
(define b2 (solid:block (position 5 0 0)
    (position 10 5 10)))
;; b2
(define b3 (solid:block (position 15 0 0)
    (position 20 5 10)))
;; b3
(define subtractb2 (solid:subtract blank b2))
;; subtractb2
(define subtractb3 (solid:subtract blank b3))
;; subtractb3
(define tool (solid:cylinder
    (position –5 2.5 5) (position 30 2.5 5)1))
;; tool
(define g (bool:select1 blank tool))
;; g
; Find the cell entities of the blank.
(define cells (entity:cells blank))
;; cells
; Create a list of cells to keep.
; Highlight the portion we’re going to throw away.
(define highlight (entity:set–highlight
    (list–ref cells 6) #t))
;; highlight
; #[entity 12 1]
; Create the list of cells we’re going to keep.
(define keep–list (list
    (list–ref cells 0)
    (list–ref cells 1)
    (list–ref cells 2)
    (list–ref cells 3)
    (list–ref cells 4)
    (list–ref cells 5)
    (list–ref cells 7)))
;; keep–list
(define select (bool:select2 blank keep–list))
;; select
```

entity:edges

Scheme Extension: Cellular Topology

Action: Returns list of all edge entities in an entity or list of entities.

Filename: ct/ct_scm/cell_scm.cxx

APIs: api_get_edges_from_all_entities

Syntax: `(entity:edges entity–list)`

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

Returns: edge | edge ...

Errors: None

Description: Returns a list of the edges for the input entity or entity–list. Returns an empty list when no edges are found.

Limitations: None

Example:

```
; entity:edges
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 25 25)))
;; block1
; List the block's edges.
(entity:edges block1)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;; #[entity 6 1] #[entity 7 1] #[entity 8 1]
;; #[entity 9 1] #[entity 10 1] #[entity 11 1]
;; #[entity 12 1] #[entity 13 1] #[entity 14 1])
```

entity:faces

Scheme Extension: Cellular Topology

Action: Returns list of all face entities for an entity or list of entities.

Filename: ct/ct_scm/cell_scm.cxx

APIs: api_get_faces_from_all_entities

Syntax: `(entity:faces entity–list)`

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

Returns: face | face ...

Errors: None

Description: Returns an empty list when no faces are found.

Input argument is an entity–list whose list of all faces is to be obtained.

Limitations: None

Example:
```
; entity:faces
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 15 5)))
;; block1
; List the block's faces.
(entity:faces block1)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;; #[entity 6 1] #[entity 7 1] #[entity 8 1])
```

entity:vertices

Scheme Extension: Cellular Topology

Action: Returns list of all vertices in an entity or list of entities.

Filename: ct/ct_scm/cell_scm.cxx

APIs: api_get_vertices_from_all_entities

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

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

Returns: vertex | vertex ...

Errors: None

Description: Returns a list of the vertices for the input entity or entity–list. Returns an empty list when no vertices are found.

Limitations: None

Example:

```
; entity:vertices
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 25 25 25)))
;; block1
; List the block's edges.
(entity:vertices block1)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;; #[entity 6 1] #[entity 7 1] #[entity 8 1]
;; #[entity 9 1] #[entity 10 1])
```