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.

curve:intersect

Scheme Extension: Intersectors

Action: Gets the intersection between two edges or curves.

Filename: intr/intr_scm/icrv_scm.cxx

APIs: api_intersect_curves

Syntax:
```
(curve:intersect curve1 curve2 [bounded=#t]
[acis-opts])
```

Arg Types:

curve1	curve | edge
curve2	curve | edge
bounded	boolean
acis–opts	acis–options

Returns: (position ...)

Errors: EDGE with no CURVE

Description: curve1 specifies an edge or a curve.

curve2 specifies a curve.

If bounded is #t, this extension only reports intersections within curve bounds.

The optional argument acis–opts helps enable journaling and versioning options.

This extension returns an empty list if the curves do not intersect. It returns a list of positions where the input curves intersect if there is a finite number of intersection points. If the boolean bounded is #t, the default, and if the curves share a portion of the underlying geometry, then this extension returns a list containing the endpoints of the shared portion. If the boolean bounded is #f, and if the curves share a portion of the underlying geometry, then this extension returns a list of the endpoints of the shared portion.

The bounded geometry of a spline edge and the fully underlying geometry of spline edges coincide. Thus, when bounded is #f, extrapolated extensions of spline edges does not occur.

Limitations: The acis_options object only applies to api_intersect_curves.

Example:

```
; curve:intersect
; Create circular edge 1.
(define edge1
    (edge:circular (position 0 0 0) 25 0 185))
;; edge1
; Create circular edge 2.
(define edge2 (edge:circular
    (position 0 10 0) 20 –180))
;; edge2
; Determine the intersections of the two edges.
(curve:intersect edge1 edge2)
;; (#[position –18.9983551919633 16.25 0]
;; #[position 18.9983551919633 16.25 0])

; Additional example.
(define line1 (edge:linear (position 0 5 0)
    (position 0 –40 0)))
;; line1
(define line2 (edge:linear (position –40 –10 0)
    (position 20 5 0)))
;; line2
; find point of intersection between lines and
; but only intersections within bounds
(curve:intersect line1 line2)
;; (#[position 0 4.44089209850063e–16 0])
(curve:intersect line1 line2 #f)
;; (#[position 0 4.44089209850063e–16 0])
```

cvty:concave

Scheme Extension: Model Geometry, Model Topology

Action: Determines whether a given cvty is concave or not.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**cvty:concave** cvty)

Arg Types: cvty scm_cvty

Returns: boolean

Errors: None

Description: This extension returns #t if the given cvty is concave and #f if it is not. Equivalent to the C++ cvty::concave.

Note *Additional enquiry functions such as* cvty:concave–tangent *etc. are defined in scm/examples/cvty.scm*

cvty represents the convexity at a point or along a single edge (or something equivalent), such as convex, tangent convex etc.

Limitations: None

Example:

```
; cvty:concave
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
; Determine if an edge is concave
(cvty:concave (pt–cvty–info:instantiate
    (edge:mid–pt–cvty–info
    (list–ref edge–list 0)) –1))
;; #f
```

cvty:convex

Scheme Extension: Model Geometry, Model Topology

Action: Determines whether a given cvty is convex or not.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**cvty:convex** cvty)

Arg Types: cvty scm_cvty

Returns: boolean

Errors: None

Description: This extension returns #t if the given cvty is convex and #f if it is not. Equivalent to the C++ cvty::convex.

Note *Additional enquiry functions such as* cvty:convex–tangent *etc. are defined in scm/examples/cvty.scm*

cvty represents the convexity at a point or along a single edge (or something equivalent), such as convex, tangent convex etc.

Limitations: None

Example:

```
; cvty:convex
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
; Determine if an edge is convex
(cvty:convex (pt–cvty–info:instantiate
    (edge:mid–pt–cvty–info
    (list–ref edge–list 0)) –1))
;; #t
```

cvty:inflect

Scheme Extension: Model Geometry, Model Topology

Action: Determines whether the given cvty is inflected or not.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**cvty:inflect** cvty)

Arg Types: cvty scm_cvty

Returns: boolean

Errors: None

Description: Returns #t if the given cvty is inflected and #f if not. Equivalent to the C++ cvty::inflect.

Note *Additional enquiry functions such as* cvty:tangent–inflect *etc. are defined in scm/examples/cvty.scm*

cvty represents the convexity at a point or along a single edge (or something equivalent), such as convex, tangent convex etc.

Limitations: None

Example:

```
; cvty:inflect
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(cvty:inflect (pt–cvty–info:instantiate
    (edge:mid–pt–cvty–info
    (list–ref edge–list 0)) –1))
;; #f
```

cvty:knife

Scheme Extension: Model Geometry, Model Topology

Action: Determines whether the given cvty is of knife convexity or not.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**cvty:knife** cvty)

Arg Types: cvty scm_cvty

Returns: boolean

Errors: None

Description: Returns #t if the given cvty is knife and #f if not. Equivalent to the C++ cvty::knife etc.

Note *Additional enquiry functions such as* cvty:knife–convex *etc. are defined in scm/examples/cvty.scm*

cvty represents the convexity at a point or along a single edge (or something equivalent), such as convex, tangent convex etc.

Limitations: None

Example:

```
; cvty:knife
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(cvty:knife (pt–cvty–info:instantiate
    (edge:mid–pt–cvty–info
    (list–ref edge–list 0)) –1))
;; #f
```

cvty:mixed

Scheme Extension: Model Geometry, Model Topology

Action: Determines whether the given cvty is of mixed convexity or not.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**cvty:mixed** cvty)

Arg Types:	cvty	scm_cvty

Returns: boolean

Errors: None

Description: Returns #t if the given cvty is of mixed convexity and #f if not. Equivalent to the C++ cvty::mixed etc.

Note *Additional enquiry functions such as* cvty:tangent–mixed *etc. are defined in scm/examples/cvty.scm*

cvty represents the convexity at a point or along a single edge (or something equivalent), such as convex, tangent convex etc.

Limitations: None

Example:
```
; cvty:mixed
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge-list (entity:edges block1))
;; edge-list
(cvty:mixed (pt-cvty-info:instantiate
    (edge:mid-pt-cvty-info
    (list-ref edge-list 0)) -1))
;; #f
```

cvty:tangent

Scheme Extension: Model Geometry, Model Topology

Action: Determines whether the given cvty is tangent or not.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**cvty:tangent** cvty)

Arg Types:	cvty	scm_cvty

Returns: boolean

Errors: None

Description: Returns #t if the given cvty is tangent and #f if not. Equivalent to the C++ cvty::tangent etc.

cvty represents the convexity at a point or along a single edge (or something equivalent), such as convex, tangent convex etc.

Note *Additional enquiry functions such as* cvty:concave–tangent *etc. are defined in scm/examples/cvty.scm*

Limitations: None

Example:

```
; cvty:tangent
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge-list (entity:edges block1))
;; edge-list
(cvty:tangent (pt-cvty-info:instantiate
    (edge:mid-pt-cvty-info
    (list-ref edge-list 0)) -1))
;; #f
```

cvty:unknown

Scheme Extension: Model Geometry, Model Topology

Action: Determines whether this cvty is unknown or not.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**cvty:unknown** cvty)

Arg Types: cvty scm_cvty

Returns: boolean

Errors: None

Description: Returns #t if the given cvty is unknown and #f if not. Equivalent to the C++ cvty::unknown etc.

cvty represents the convexity at a point or along a single edge (or something equivalent), such as convex, tangent convex etc.

Limitations: None

Example:

```
; cvty:unknown
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge-list (entity:edges block1))
;; edge-list
(cvty:unknown (pt-cvty-info:instantiate
    (edge:mid-pt-cvty-info
    (list-ref edge-list 0)) -1))
;; #f
```

cvty:unset

Scheme Extension: Model Geometry, Model Topology

Action: Determines whether this cvty is unset or not.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**cvty:unset** cvty)

Arg Types: cvty scm_cvty

Returns: boolean

Errors: None

Description: Returns #t if the given cvty is unset and #f if not. Equivalent to the C++ cvty::unset .

cvty represents the convexity at a point or along a single edge (or something equivalent), such as convex, tangent convex etc.

Limitations: None

Example:

```
; cvty:unset
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(cvty:unset (pt–cvty–info:instantiate
    (edge:mid–pt–cvty–info
    (list–ref edge–list 0)) –1))
;; #f
```

ed–cvty–info:angles

Scheme Extension: Model Geometry, Model Topology

Action: Returns the angles recorded in an ed-cvty-info attribute.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: `(ed–cvty–info:angles ed–cvty–info)`

Arg Types:

ed–cvty–info	scm_ed_cvty_info

Returns: (real . real)

Errors: None

Description: Returns the maximum and minimum signed angles between face normals along an edge, as stored in the ed-cvty-info. Negative numbers imply concave, positive numbers imply convex. Equivalent to the C++ ed_cvty_info::angles.

tolerance specifies the angle tolerance.

Limitations: None

Example:

```
; ed–cvty–info:angles
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
; Get recorded angles.
(ed–cvty–info:angles (edge:ed–cvty–info
    (list–ref edge–list 0)))
;; (1 . 1)
```

ed–cvty–info:instantiate

Scheme Extension: Model Geometry, Model Topology

Action: Instantiate the given cvty convexity information to a given tolerance.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: `(ed–cvty–info:instantiate ed–cvty–info tolerance)`

Arg Types:

ed–cvty–info	scm_ed_cvty_info
tolerance	real

Returns: scm_cvty

Errors: None

Description: Based on the given angle tolerance for determining the limit for ’smoothness’, this extension determines the convexity of the edge for which the ed-cvty-info was computed. Equivalent to the C++ ed_cvty_info::instantiate.

ed-cvty-info represents the convexity of an edge.

tolerance specifies the angle tolerance.

Limitations: None

Example:

```
; ed–cvty–info:instantiate
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(ed–cvty–info:instantiate (edge:ed–cvty–info
    (list–ref edge–list 0)) .01)
;; #[cvty: cvx]
```

ed–cvty–info:tangent–convexity

Scheme Extension: Model Geometry, Model Topology

Action: Returns the tangent convexity of an ed–cvty–info.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**ed-cvty-info:tangent-convexity** ed–cvty–info)

Arg Types:	ed–cvty–info	scm_ed_cvty_info

Returns: scm_ed_cvty_info

Errors: None

Description: This extension returns the convexity of an ed–cvty–info if that edge was viewed with an angle tolerance large enough for it to be considered a smooth (tangent) edge. Entirely equivalent to the C++ ed_cvty_info::tangent_convexity.

ed-cvty-info represents the convexity of an edge.

Limitations: None

Example:
```
; ed-cvty-info:tangent-convexity
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(ed–cvty–info:tangent–convexity
    (edge:ed–cvty–info (list–ref edge–list 0)))
;; #[cvty: knf]
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(ed–cvty–info:tangent–convexity
    (edge:ed–cvty–info (list–ref edge–list 0)))
;; #[cvty: knf]
```

ed–cvty–info:unknown

Scheme Extension: Model Geometry, Model Topology

Action: Returns whether the convexity of an ed-cvty-info is unknown.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**ed–cvty–info:unknown** ed–cvty–info)

Arg Types:	ed–cvty–info	scm_ed_cvty_info

Returns: boolean

Errors: None

Description: Returns #t if the convexity computed in an ed-cvty-info is unknown. Otherwise, it returns #f. Equivalent to the C++ ed_cvty_info::unknown.

ed-cvty-info represents the convexity of an edge.

Limitations: None

Example:

```
; ed–cvty–info:unknown
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(ed–cvty–info:unknown (edge:ed–cvty–info
    (list–ref edge–list 0)))
;; #f
```

ed–cvty–info:unset

Scheme Extension: Model Geometry, Model Topology

Action: Returns whether the convexity of an ed-cvty-info is unset.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**ed–cvty–info:unset** ed–cvty–info)

Arg Types: ed–cvty–info scm_ed_cvty_info

Returns: boolean

Errors: None

Description: Returns #t if the convexity computed in an ed-cvty-info is unset. Otherwise it returns #f. Equivalent to the C++ ed_cvty_info::unset. It should not actually be possible to create an ”unset” ed-cvty-info, but this function is provided for completeness.

ed-cvty-info represents the convexity of an edge.

Limitations: None

Example:

```
; ed–cvty–info:unset
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
; Find out whether convexity is unset.
(ed–cvty–info:unset (edge:ed–cvty–info
    (list–ref edge–list 0)))
;; #f
```

edge:convexity

Scheme Extension: Model Topology

Action: Determines the convexity of an edge.

Filename: intr/intr_scm/chk_scm.cxx

APIs: api_edge_convexity_param

Syntax: `(edge:convexity edge [param] [acis–opts])`

Arg Types:

edge	edge
param	real
acis–opts	acis–options

Returns: string

Errors: None

Description: This extension returns the convexity of an edge. If param is supplied, the convexity at that point is determined. Otherwise, convexity is determined at the mid–point of the edge.

The optional argument acis–opts helps enable journaling and versioning options.

Limitations: None

Example:

```
; edge:convexity
; Create a couple blocks for example.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
(define block2 (solid:block (position 0 10 0)
    (position 20 20 10)))
;; block2
; Unite the two blocks
(define unite (solid:unite block1 block2))
;; unite
; Find the edges of blocks 1 and 2
(define edgelist (entity:edges unite))
;; edgelist
; Define edges 1 and 2
(define edge1 (list–ref edgelist 0))
;; edge1
; verify edge1 is concave.
(edge:convexity edge1)
;; ”concave”
(define edge2 (list–ref edgelist 17))
;; edge2
(edge:convexity edge2)
;; ”convex”
; OUTPUT convexity
```

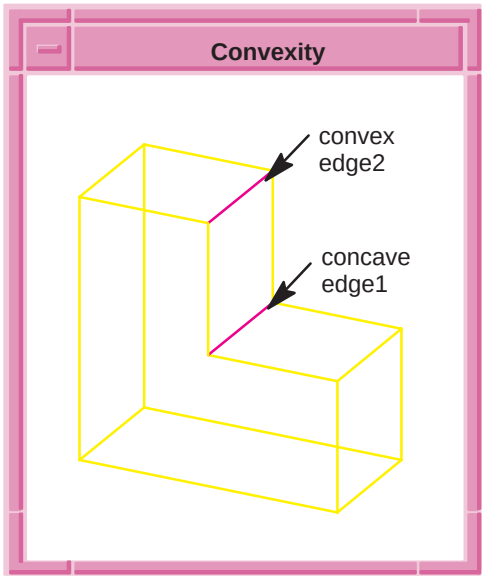

Figure 2-1. edge:convexity

edge:ed–cvty–info

Scheme Extension: Model Topology

Action: Computes convexity information for a whole edge, or part of an edge.

Filename: intr/intr_scm/cvty_scm.cxx

APIs: None

Syntax:
```
(edge:ed-cvty-info edge [use-curvatures=#t]
    [approx-ok=#f] [interval=entire common-range])
```

Arg Types:		
	edge	edge
	use–curvatures	boolean
	approx–ok	boolean
	interval	real . real
	common–range	real

Returns: scm_ed_cvty_info

Errors: None

Description: Compute convexity information for an entire edge, or part thereof. This includes the [sine of] the minimum and maximum angles along the edge (+ve for convex, –ve for concave). This also applies to the convexity of this edge if viewed with an angle tolerance large enough to regard it as ”flat”).

Limitations: None

Example:

```
; edge:ed–cvty–info
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(define edge1 (list–ref edge–list 0))
;; edge1
; Get information on an edge for a 0.01 angle
;   tolerance.
(ed–cvty–info:instantiate (edge:ed–cvty–info
    edge1) 0.01)
;; #[cvty: cvx]
```

edge:end–pt–cvty–info

Scheme Extension: Model Topology

Action: Computes convexity information at end point of edge.

Filename: intr/intr_scm/cvty_scm.cxx

APIs: None

Syntax: (**edge:end–pt–cvty–info** edge [use–curvatures=#t])

Arg Types:	edge	edge
	use–curvatures	boolean

Returns: scm_pt_cvty_info

Errors: None

Description: Compute convexity information at end point along an edge. This consists of an angle (the [sine of the] angle between surface normals there, +ve meaning convex), and the convexity of the edge here if viewed with an angle tolerance sufficiently large that we would regard this point as tangent. Note that for tolerant edges, the “common range” of the edge is used, which may differ slightly. This is the range over which the nominal edge curve and the coedge 3D curves appear to overlap “sensibly”.

Limitations: None

Example:
```
; edge:end–pt–cvty–info
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(edge:end–pt–cvty–info (list–ref edge–list 0))
;; #[pt_cvty_info: 1 [cvty: knf] (tol 1e–10)]
```

edge:entity–rel

Scheme Extension: Object Relationships

Action: Returns the relationship between an edge and another entity (point, edge, face or body).

Filename: intr/intr_scm/edent_scm.cxx

APIs: api_edent_rel

Syntax: (**edge:entity–rel** edge entity)

Arg Types:	edge	edge
	entity	entity

Returns: (string (position position...) (pair pair...))

Errors: None

Description: Determines the relationships between the given edge and the given entity that may be a point, edge, face, or body.

If no relationship exists between the edge and the entity, then an empty list is returned. If a relationship is returned, a list is returned with three entries. The first entry is a string describing the relationship, the second is a list of intersection points found between the edge and the entity, and the third is a list of the changes in relationship as the edge passes through the corresponding intersection points.

A point may be on or off the edge. In this case, the first entry in the returned list will be either ”point on edge” or ”point off edge”. If the point is on the edge, the second entry will be a list containing the position of the point, otherwise it will be an empty list. The relations list is always empty for point–edge relations, as there is no sense in which the change in relationship can be described.

An edge may intersect, overlap, or be identical to the edge. The edge may lie in the surface of a face and be completely inside, partially inside, or on the boundary of the face, or it may simply intersect the face. Likewise, an edge may be inside, partially inside, or on the boundary of a body. The notion of being inside/outside is meaningful only if the boundary bounds a finite volume.

Limitations: None

Example:

```
; edge:entity–rel
; Define an edge
(define e1 (edge:linear (position 0 0 0)
    (position 20 0 0)))
;; e1
; Define another edge
(define e2 (edge:linear (position 10 0 0)
    (position 30 0 0)))
;; e2
(edge:entity–rel e1 e2)
;; (”overlapping edges” (#[position 10 0 0]
;; #[position 20 0 0]) ((”tangent” . ”coincident”)
;; (”coincident” . ”unknown”)))
```

edge:mid–pt–cvty–info

Scheme Extension: Model Topology

Action: Compute convexity information at mid–point of edge.

Filename: intr/intr_scm/cvty_scm.cxx

APIs: None

Syntax: (**edge:mid–pt–cvty–info** edge [use–curvatures=#t])

Arg Types:	edge	edge
	use–curvatures	boolean

Returns: scm_pt_cvty_info

Errors: None

Description: Compute convexity information at a midpoint along an edge. This consists of an angle (the [sine of the] angle between surface normals there, +ve meaning convex), and the convexity of the edge here if viewed with an angle tolerance sufficiently large that we would regard this point as tangent. Note that for tolerant edges, the *common range* of the edge is used, which may differ slightly. This is the range over which the nominal edge curve and the coedge 3D curves appear to overlap ”sensibly”.

Limitations: None

Example:
```
; edge:mid–pt–cvty–info
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(edge:mid–pt–cvty–info (list–ref edge–list 0))
;; #[pt_cvty_info: 1 [cvty: knf] (tol 1e–10)]
```

edge:pt–cvty–info

Scheme Extension: Model Topology

Action: Compute convexity information at a point on an edge.

Filename: intr/intr_scm/cvty_scm.cxx

APIs: None

Syntax:
```
(edge:pt–cvty–info edge param
    [use–curvatures=#t])
```

Arg Types:		
	edge	edge
	param	real
	use–curvatures	boolean

Returns: scm_pt_cvty_info

Errors: None

Description: Compute convexity information at a specified parameter point (start–point) along an edge. This consists of an angle (the [sine of the] angle between surface normals there, +ve meaning convex), and the convexity of the edge here if viewed with an angle tolerance sufficiently large that we would regard this point as tangent.

Limitations: None

Example:

```
; edge:pt-cvty-info
; Create a block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge-list (entity:edges block1))
;; edge-list
(edge:pt-cvty-info (list-ref edge-list 0) 1)
;; #[pt_cvty_info: 1 [cvty: knf] (tol 1e-10)]
```

edge:split–at–proportion

Scheme Extension: Model Topology

Action: Splits an edge with a new vertex at the defined proportion of the way along the edge.

Filename: intr/intr_scm/icrv_scm.cxx

APIs: None

Syntax: (**edge:split-at-proportion** edge proportion)

Arg Types:		
	edge	edge
	proportion	real

Returns: vertex

Errors: None

Description: This extension is provided to test sg_split_edge_at_vertex.

proportion argument takes a real range between 0 and 1.

Limitations: None

Example:

```
; edge:split-at-proportion
; create solid block.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define split (edge:split-at-proportion
    (car (entity:edges b)) 0.5))
;; split
```

edge:start–pt–cvty–info

Scheme Extension: Model Topology

Action: Compute convexity information at start point of edge.

Filename: intr/intr_scm/cvty_scm.cxx

APIs: None

Syntax: `(edge:start-pt-cvty-info edge [use-curvatures=#t])`

Arg Types:

edge	edge
use–curvatures	boolean

Returns: scm_pt_cvty_info

Errors: None

Description: Compute convexity information at a start point along an edge. This consists of an angle (the [sine of the] angle between surface normals there, +ve meaning convex), This also applies to the convexity of this edge if viewed with an angle tolerance large enough to regard it as ”tangent”). For tolerant edges, the *common range* of the edge is used and may differ slightly. This is the range over which the nominal edge curve and the coedge 3D curves appear to overlap ”sensibly”.

Limitations: None

Example:

```
; edge:start-pt-cvty-info
; create a block
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge-list (entity:edges block1))
;; edge-list
(edge:start-pt-cvty-info (list-ref edge-list 0))
;; #[pt_cvty_info: 1 [cvty: knf] (tol 1e-10)]
```

entity:check

Scheme Extension: Entity, Debugging

Action: Checks the data structure, topology, and geometry on a single or list of entities.

Filename: intr/intr_scm/chk_scm.cxx

APIs: api_check_entity

Syntax:

```
(entity:check entity-list [check-level]
    [insanity-type] [acis-opts])
```

Arg Types:	entity–list	entity | (entity ...)
	check–level	integer
	insanity–type	string
	acis–opts	acis–options

Returns: entity | (entity ...)

Errors: None

Description: This extension performs data structure, topological, and geometric checks on an entity or list of entities. The results are printed to the debug file, which can be set with the debug:file extension.

If check–level is specified, the check is done at that level, then returned to its previous value when finished. If the check–level defined is not a multiple of ten, it is treated as the next lowest valid number. Any number over 70 is equal to 70.

The valid check levels are:

10 = Fast error checks
20 = Level 10 checks plus slower error checks
30 = Level 20 checks plus D–Cubed curve and surface checks
40 = Level 30 checks plus fast warning checks
50 = Level 40 checks plus slower warning checks
60 = Level 50 checks plus slow edge convexity change point checks
70 = Level 60 checks plus face/face intersection checks

If the check_level is not a multiple of ten, it has the same effect as the next lowest valid number. Any number higher than 70 is equal to 70.

With the optional insanity–type only the specified insanities are printed to the debug file. The option "check_abort" becomes invalid when insanity–type is specified. The options are "ERROR", "WARNING', or "NOTE".

The specific tests that entity:check performs include:

Data Structure Checks

– The parent has appropriate child–level entity; e.g., body has lump.
– Presence (non–NULL) and closure of backptr from child to parent; e.g., body's lump points to body.
– The coedge on spline surface has PCURVE.
– PCURVE indexing (0/+–1/+–2) is appropriate.
– The PCURVE has non–NULL bs2_curve.
– If EDGE has non–NULL CURVE, then CURVE must have equation.

Topological Checks

- LOOPS must be closed in both the next and previous directions.
- Apex edge loops are correct.
- COEDGE has a partner, except apex COEDGE
- All COEDGE partners point to same EDGE
- Sequential COEDGEs share a VERTEX
- EDGE is in exactly one of start and end VERTEXs edge groups.
  For example, EDGE can be reached for 1 value of i
  using start()–>edge(i)–>coedge() and
  partner and next (or previous) pointers.

Geometric Checks

- Entities with geometry, must have non–NULL geometry.
  For example, a FACE points to a SURFACE.
- Analytic surfaces have valid definitions
  a. plane must have a unit vector normal
  b. sphere must have a non–zero radius
  c. cone has:
     1. unit vector normal
     2. non–zero length major axis
     3. normal and major axis are perpendicular
     4. 0 < ratio < 1
     5. sin_angle^2 + cos_angle^2 = 1
  d. torus has
     1. unit vector normal
     2. major radius not equal to 0
     3. minor radius not equal to 0
     4. major radius >= –fabs(minor radius)

- PCURVE surface matches FACE surface (warning only if
  not equal since surface could be trimmed).
- PCURVE form is agrees with CURVE form,
  e.g. closed, open, periodic.
- PCURVE parameter period agrees with curve period.
- PCURVE at points 0, 1/3, 2/3, and 1 way along curve
  must lie on the EDGE and tangent directions at these points
  must roughly agree, i.e., have positive dot product.
  This also tests the following:
  a. PCURVE and EDGE geometry direction agree (up to sense),
  b. Start and end parameters of PCURVE
     match those of COEDGE, and
  c. Start and end locations of PCURVE (wrt to surface)
     match those of COEDGE

- Spline SURFACE form is set correctly,
  e.g. SURFACEs closed in u report this.
  Checks the underlying bs3 surface at 10 points
  along seam to verify form.
- Checks that COEDGE vertices do not lie on
  spline SURFACE singularities.
- FACE normal is consistent with COEDGE direction, i.e.,
  a. Face area is greater than 0, and
  b. Multi–LOOP FACEs have LOOPs correctly oriented.
  c. Checks that all LOOPs contain the others.

- D3 surface checks (turned on by an option).
  Checks for a bad degeneracy – adjacent degenerate boundaries.
  The remaining checks are just on spline surfaces:
  a. surface irregular
  b. self intersection
  c. closure wrong
  d. no bs3 surface
  e. control point coincidence
  f. not C1, G0, G1, or G2

- Start and end VERTEXs of COEDGE lie on FACE
- EDGE lies on FACE. Checks at 10 points along EDGE
- Start and end VERTEXs lie on EDGE geometry.
- FACEs are ordered correctly around EDGE, according to sidedness.
- COEDGES are ordered correctly around EDGE,
  according to FACE curvature
- EDGE has same sense as CURVE (taking reverse bit into account)
- Checks CURVE has correct form:
  a. straight has unit vector direction and param scale > 0
  b. ellipse has
     1. unit vector normal
     2. axis with length > 0
     3. normal perpendicular to axis
     4. radius > 0
     5. ratio < 1
  c. intcurve is correctly labeled open, closed, or periodic

- EDGE parameter range is good and agrees with start and end points.
- Check EDGE for bad approximation direction
- D3 checks on intcurve (option that can be turned on)
  a. has bs3_curve
  b. closure is correctly reported
  c. continuity is C0, G1, or G2
  d. no coincident control points
  e. curve is not irregular
  f. no self intersections
  g. no degeneracy
  h. no untreatable singularity
- No two VERTEXs have the same location.

- Optional FACE/FACE intersection checking (option check_ff_int)
  a. Two valid FACEs have proper intersection
     1. adjacent faces intersect only in along common edge
     2. non–adjacent faces do not intersect
  b. When a FACE/FACE boolean fails, checks for proper EDGE/EDGE intersections on each FACE
     1. adjacent EDGEs intersection in common VERTEX only
     2. non–adjacent EDGEs do not intersect
  c. Two valid, non–intersecting SHELLs in the same LUMP have proper containment, i.e., each SHELL contains the other.
     Note, a SHELL is valid if it contains no improperly intersecting FACEs.
  d. Two valid, non–intersecting LUMPs have proper containment, i.e., neither LUMP contains the other.
     Note, a LUMP is valid when it contains no improperly intersecting FACEs and no SHELLs with improper containment.

When FACE/FACE problems are found, a BODY containing the bad intersections is created. This BODY is named check_error.

The optional argument acis–opts helps enable journaling and versioning options.

Limitations: None

Example:

```
; entity:check
; Create solid block 1.
(define block1 (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block1
; Determine if the entity passes the checks.
(entity:check block1)
; checked:
;   1 lumps
;   1 shells
;   0 wires
;   6 faces
;   6 loops
;   24 coedges
;   12 edges
;   8 vertices
;; ()
; Create a circular edge.
(define edge1
    (edge:circular (position 0 0 0) 25 0 185))
;; edge1
; Determine if edge1 passes the checks.
(entity:check edge1)
; entid 1236816: edge without backptr
; entid 1231288: vertex has edge 1236816
; in group 0 times
; entid 1231312: vertex has edge 1236816
; in group 0 times
; checked:
;   0 lumps
;   0 shells
;   0 wires
;   0 faces
;   0 loops
;   0 coedges
;   1 edges
;   2 vertices
;; (#[entity 3 1])
```

entity:dist

Scheme Extension: Laws, Object Relationships

Action: Gets the minimum distance between two entities or entity and a position.

Filename: intr/intr_scm/ilaw_scm.cxx

APIs: api_entity_entity_distance, api_entity_point_distance

Syntax: `(entity:dist part1 part2 [acis–opts])`

Arg Types:

part1	entity | position
part2	entity | position
acis–opts	acis–options

Returns: (position . (entity | entity ...) . string)

Errors: None

Description: Using the two input entities, this finds a position on each entity such that the distance between the two is the minimum distance. Supported entities include VERTEX, EDGE, LOOP, FACE, WIRE, SHELL, LUMP, and BODY. The command can also find the minimum distance using an entity and a position, or a position and an entity.

The optional argument acis–opts helps enable journaling and versioning options.

Limitations: If part1 is defined as a position, part2 must be an entity.

Example:

```
; entity:dist
; Create a law surface and lemon torus.
(define c1 (face:law "vec
    (x,y,sin (x)*cos (y))" –10 10 –10 10))
;; c1
(define c2 (face:torus
    (position 0 0 9.5) –5 10 0 360 0 360
    (gvector 0 0 –1)))
;; c2
; Get minimum distance between faces
(entity:dist c1 c2)
;; (0.60223495915902 #[position 0.407311827177182 0
;; 0.396142511453466] #[par–pos 0.407311827177182
;; 0] #[entity 2 1] "FACE" #[position
;; 8.88178419700125e–016 1.08766893609269e–031
;; 0.839745962155614] 0 #[entity 4 1] "VERTEX")
```

entity:extrema

Scheme Extension: Model Geometry, Analyzing Models

Action: Finds an extreme position on an entity.

Filename: intr/intr_scm/rtst_scm.cxx

APIs: api_entity_extrema

Syntax: (**entity:extrema** entity vector1 [vector2] [vector3] [acis–opts])

Arg Types:		
	entity	entity
	vector1	position
	vector2	position
	vector3	position
	acis–opts	acis–options

Returns: ((entity | (entity ...)) . position)

Errors: None

Description: vector1, vector2 and vector3 are positions in the entity.

The optional argument acis–opts helps enable journaling and versioning options.

Limitations: None

Example:

```
; entity:extrema
; create topology to illustrate command.
(define s1 (solid:block
    (position 0 0 0) (position 10 10 5)))
;; s1
(define s2 (solid:block
    (position 3 0 5) (position 7 3 8)))
;; s2
(define s3 (solid:block
    (position 3 7 5) (position 7 10 2)))
;; s3
(define unite (bool:unite s1 s2))
;; unite
(define subtract (bool:subtract s1 s3))
;; subtract
(define extrema (entity:extrema s1 (gvector 1 1 1)
    (gvector 0 0 1) (gvector 1 0 0)))
;; extrema
```

entity:pattern

Scheme Extension: Patterns

Action: Creates a pattern of entities from a seed entity.

Filename: intr/intr_scm/ilaw_scm.cxx

APIs: api_set_entity_pattern

Syntax:
```
(entity:pattern entity pattern [copy–pat
    [seed–index [no–cross–face–list] [check]]]
[acis–opts])
```

Arg Types:		
	entity	entity
	pattern	pattern
	copy–pat	boolean
	seed–index	integer
	no–cross–face–list	entity | (entity ...)
	check	boolean
	acis–opts	acis–options

Returns: entity

Errors: None

Description: This Scheme extension applies the pattern to the seed entity entity. By default, a copy of the pattern is made, and it is the copy that is actually applied to the entity. This behavior can be overridden by setting copy–pat to FALSE. However, when copying is overridden and in–pat is shared by multiple bodies, a transform placed upon the bodies is transferred to the pattern multiple times, which is clearly undesirable. Also by default, ent is associated with the first pattern element (index 0), but may be associated with another element by furnishing the associated zero–based seed–index.

For cases in which the pattern is applied to a “bump” on a substrate rather than to an autonomous entity, the limits of the bump are automatically computed, but the user may choose to override the default limits by furnishing a list of no–cross–faces.

For performance reasons, the extension does not check the generated pattern of entities for intersection, containment, or compatibility unless the user sets the flag check to TRUE.

The optional argument acis–opts helps enable journaling and versioning options.

Limitations: None

Example:

```
; entity:pattern
; Make a spherical body.
(define body (solid:sphere (position 0 0 0) 1))
;; body
; Make a circular pattern.
(define center (position 11 2 3))
;; center
(define normal (gvector 0 0 1))
;; normal
(define num 12)
;; num
(define pat (pattern:elliptical center normal num))
;; pat
; Apply the pattern to the body.
(define new–pattern (entity:pattern body pat))
;; new–pattern
```

entity:point–distance

Scheme Extension: Object Relationships, Intersectors

Action: Returns the minimum distance between an entity and a position.

Filename: intr/intr_scm/ilaw_scm.cxx

APIs: api_entity_point_distance

Syntax: (**entity:point–distance** entity position [acis–opts])

Arg Types:

entity	entity
position	position
acis–opts	acis–options

Returns: real

Errors: None

Description: This extension finds the distance between an entity and a position. The closest point on the entity is also returned.

The optional argument acis–opts helps enable journaling and versioning options.

Limitations: None

Example:

```
; entity:point–distance
; Create a block and get the distance
; between a given point and the block.
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
(define distance1 (entity:point–distance
    block1 (position 30 25 37)))
; Point on entity 10.000000 20.000000 20.000000
;; distance1
```

entity:touch

Scheme Extension: Intersectors

Action: Determines whether the two defined entities (or any two entities in the lists) “touch” (distance < 2 * resabs).

Filename: intr/intr_scm/ilaw_scm.cxx

APIs: api_entity_entity_touch

Syntax:

```
(entity:touch {entity1 | entity–list1}
    {entity2 | entity–list1 | entity–list2}
[acis–opts])
```

Arg Types:		
	entity1	entity
	entity2	entity
	entity–list1	entity | (entity...)
	entity–list2	entity | (entity...)
	acis–opts	acis–options

Returns: boolean

Errors: None

Description: Using the two input entities or entity lists, determines whether the entities touch each other or if any entity in list 1 touches any entity in list 2. Supported entities include VERTEX, EDGE, LOOP, FACE, WIRE, SHELL, LUMP, and BODY.

The optional argument acis–opts helps enable journaling and versioning options.

Limitations: None

Example:

```
; entity:touch
; Create four cylinders.
(define c1 (face:cylinder (position 0 0 0)
    (position 0 0 1) 1 1))
;; c1
(define c2 (face:cylinder (position 1 0 0)
    (position 1 0 1) 1 1))
;; c2
(define c3 (face:cylinder (position 2 5 0)
    (position 2 5 1) 1 1))
;; c3
(define c4 (face:cylinder (position 3 5 0)
    (position 3 5 1) 1 1))
;; c4
(zoom–all)
;; #[view 1573816]
; See if c1 touches c2
(entity:touch c1 c2)
;; #t
; See if any cylinders in the first list touch
; any in second list
(entity:touch (list c1 c2) (list c3 c4))
;; #f
```

face:get–silhouette

Scheme Extension: Silhouette and Isoparametric Curves

Action: Computes the silhouette curves on the face

Filename: intr/intr_scm/icrv_scm.cxx

APIs: api_silhouette_edges

Syntax: `(`**`face:get–silhouette`** `face [from–pos to–pos] [proj–type] [transf] [ao])`

Arg Types:	face	face
	from–pos	position
	to–pos	position
	proj–type	boolean
	transf	transf
	acis–opts	acis–options

Returns: edge ...

Errors: There must be at least one view.

Description: Computes the silhouette curves on the surface of the face and trims them to the boundaries of the face. The silhouette curves can be calculated for either a parallel or perspective projection. It retrieves a list of EDGEs representing the bounded portion of the silhouette curves within the face.

face is the face to be examined.

from–pos is the position of viewer.

to–pos helps is the position towards which user looks.

proj–type is the projection type: 0 for parallel and 1 for perspective.

transf is the transform.

acis–opts helps enable journaling and versioning options.

Limitations: None

Example:

```
; face:get-silhouette
;; Define solid and reference
(part:clear)
;; #t
(view:dl)
;; #[view 15795402]
(define s (solid:sphere (position 0 0 0) 1))
;; s
(iso)
;; #[view 15795402]
(zoom-all)
;; #[view 15795402]
(define f (list-ref (entity:faces s)0))
;; f
(face:get-silhouette f
    (position 2 2 2)(position -2 -2 -2))
;; (#[entity 3 1])
```

face:point–relationship

Scheme Extension: Object Relationships

Action: Determines the relationship of a position within a given face's surface to that face.

Filename: intr/intr_scm/ifac_scm.cxx

APIs: api_point_in_face

Syntax:

```
(face:point-relationship face-entity test-position
    [transf] ["par-pos-guess" u-guess v-guess]
    [prev-pos [prev-pos-containment]]
    [use-cache = 0 [cache-size = 10]] [acis-opts])
```

Arg Types:

face–entity	face
test-position	position
transf	transform
u–guess	double
v–guess	double
prev–pos	position
prev–pos–containment	string
use–cache	integer
cache–size	integer
acis–opts	acis–options

Returns: string

Errors: None

Description: Returns "inside", "outside", "boundary", or "unknown" to indicate the containment relationship between the face given by entity and the position given by test-position. It is presumed that the specified position is known to lie within the surface belonging to the face.

The transform object will be considered only if is specified otherwise, if the face belongs to a body, it will find the transform of that body and will input into the API. If it is desired to override the body transformation, provide an identinty transf.

The par-pos–guess argument provides additional help by providing a coordinate in parametric space for an initial guess on where the position may lie. If no argument is provided it will use the default (a null reference).

If a previous position is specified, a containment description must be provided ("inside", "outside", "boundary", or "unknown"). This position can provide help in finding the position on the face at a faster rate. It is recomended to use a position that was previously found using this function. The scheme command will use the proper api_point_in_face function if there are previous positions.

If the optional argument use–cache if supplied, is:

0, the cache is not used. Will be input as FALSE in api_point_in_face

1, the cache is used to determine point in face relationships. Will be input as TRUE

2, this function runs in a testing mode – it tries the point in face containments first without using the cache, then tries again with the cache verifying the answers remain the same. Will be input as TRUE also.

The optional argument cache–size specifies the size of the cache array points. This argument is only meaningful when use-cache is either 1 or 2.

Debug mode (cache = 2):
The function returns the containment string corresponding to the point in face answers for the point, except when run in debug mode (use–cache equal to 2), in which case the percentage improvement of the cached time to the time without using the cache is returned, i.e. (time_without – time_with) / time_without * 100.

face–entity specifies a face entity.

test-position specifies the position on the face.

transf specifies the body transformation.

u–guess specifies inital guess of the point along u.

v–guess specifies inital guess of the point along v.

prev–pos specifies the previous position.

prev–pos–containment gives containment decription previous position. It can take one of the following values: ”inside”, ”outside”, ”boundary”, or ”unknown”

use–cache specifies if caching is used. Possible values are 0, 1 or 2.

cache–size specifies size of cache array points. Size can be from 0 to 10.

acis–opts helps enable journaling and versioning options.

Limitations: None

Example:

```
; face:point–relationship
; Create a planar face.
(define face
    (face:plane (position 0 0 0) 10 20
    (gvector 0 0 1)))
;; face
; Define position 1
(define test1 (position 0 0 0))
;; test1
; Relationship between face and position 1
(face:point–relationship face test1)
;; ”boundary”
; Define position 2
(define test2 (position 5 10 0))
;; test2
; Relationship between face and position 2
(face:point–relationship face test2)
;; ”inside”
; Define position 3
(define test3 (position 15 10 0))
;; test3
; Relationship between face and position 3
(face:point–relationship face test3)
;; ”outside”
```

face:point–relationship–list

Scheme Extension: Object Relationships

Action: Determines the relationship of a LIST of positions within a given face’s surface to that face.

Filename: intr/intr_scm/ifac_scm.cxx

APIs: api_point_in_face

Syntax:

```
(face:point–relationship–list face–entity
position–list
    [transf] [”par–pos–guess” u–guess v–guess]
    [use–cache = 0 [cache–size = 10]] [acis–opts])
```

Arg Types:	face–entity	face
	position–list	position
	transf	transform
	u–guess	double
	v–guess	double
	use–cache	integer
	cache–size	integer
	acis–opts	acis–options

Returns: string

Errors: None

Description: Tests a list of positions for a given face.

The transform object will be considered only if is specified otherwise, if the face belongs to a body, it will find the transform of that body and will input into the API. If it is desired to override the body transformation, provide an identinty transf.

The par-pos–guess argument provides additional help by providing a coordinate in parametric space for an initial guess on where the position may lie. If no argument is provided it will use the default (a null reference).

If the optional argument use–cache if supplied, is:

0, the cache is not used. Will be input as FALSE in api_point_in_face

1, the cache is used to determine point in face relationships. Will be input as TRUE

2, this function runs in a testing mode – it tries the point in face containments first without using the cache, then tries again with the cache verifying the answers remain the same. Will be input as TRUE also.

The optional argument cache–size specifies the size of the cache array points. This argument is only meaningful when use-cache is either 1 or 2.

Debug mode (cache = 2): When run in debug mode (use–cache equal to 2), in which case the percentage improvement of the cached time to the time without using the cache is returned, i.e. (time_without – time_with) / time_without * 100.

face–entity specifies a face entity.

position–list specifies the list of positions.

transf specifies the body transformation.

u–guess specifies inital guess of the point along u.

v–guess specifies inital guess of the point along v.

use–cache specifies if caching is used.

cache–size specifies size of cache array points.

acis–opts helps enable journaling and versioning options.

Limitations: None

Example:

```
; face:point–relationship–list
; Create a planar face.
(define face
    (face:plane (position 0 0 0) 10 20
    (gvector 0 0 1)))
;; face
; Define position 1
(define test1 (position 0 0 0))
;; test1
; Define position 2
(define test2 (position 5 10 0))
;; test2
; Define position 3
(define test3 (position 15 10 0))
;; test2
; Make a list
(define l (list test1 test2 test3))
;; l
; Relationship between face and list not using the
cache
 (face:point–relationship–list face l)
;;
; Relationship between face and list using the cache
(face:point–relationship–list face l 1)
;;
```

face:ray–at–position

Scheme Extension: Object Relationships

Action: Gets the position on and normal to a face closest to a given position.

Filename: intr/intr_scm/ifac_scm.cxx

APIs: None

Syntax: `(face:ray–at–position face–entity test–position)`

Arg Types:

face–entity	face
test-position	position

Returns: ray

Errors: None

Description: The face normal is computed based on test-position and this extension returns it as a ray. The position on the entity face closest to the test–position is located.

face–entity specifies a face entity.

test-position specifies the position on the face.

Limitations: None

Example:

```
; face:ray–at–position
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 40 40 40)))
;; block1
; Get a list of the block's faces.
(define faces1 (entity:faces block1))
;; faces1
; Find the position and normal on the faces
; closest to the given points.
(face:ray–at–position (car (cdr faces1))
    (position 0 0 0))
;; #[ray (0 0 0) (0 0 –1)]
(face:ray–at–position (car (cdr (cdr (cdr
    (cdr (cdr faces1)))))) (position 50 100 30))
;; #[ray (40 40 30) (1 0 0)]
```

law:boundary–field

Scheme Extension: Model Geometry

Action: Creates a boundary field around the defined entity.

Filename: intr/intr_scm/ilaw_scm.cxx

APIs: api_create_boundary_field, api_get_coedges, api_get_faces, api_get_wires

Syntax: `(law:boundary–field entity–list options [acis–opts])`

Arg Types:

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

Returns: entity

Errors: None

Description:

```
The following list shows the valid options for the
second argument options.

"empty_field"
"uniform_vec_field" [(vector) optional vector]
"face_normal_field"
"face_tangent_field"
"direction control"
"reverse"
```

The optional argument acis–opts helps enable journaling and versioning options.

Limitations: None

Example:

```
; law:boundary–field
; Example not available for this release.
```

law:min–dist

Scheme Extension: Laws, Object Relationships

Action: Gets the minimum distance between faces.

Filename: intr/intr_scm/ilaw_scm.cxx

APIs: api_entity_entity_distance

Syntax: `(law:min–dist entity1 entity2)`

Arg Types:

entity1	entity
entity2	entity

Returns: real

Errors: None

Description: Finds a position on each entity such that the distance between the two is the minimum distance. Supported entities include vertex, edge, loop, face, wire, shell, lump, and body.

entity1 is the first entity that could be any of vertex, edge, loop, face, wire, shell, lump or body.

entity2 is the first entity that could be any of vertex, edge, loop, face, wire, shell, lump or body.

Limitations: None

Example:

```
; law:min–dist
; Create a law surface and lemon torus.
(define c1
    (face:law "vec(x,y,sin(x)*cos(y))"
    –10 10 –10 10))
;; c1
(define c2
    (face:torus (position 0 0 9.5)
    –5 10 0 360 0 360 (gvector 0 0 –1)))
;; c2
; Get minimum distance between faces
(define t10 (law:min–dist c1 c2))
; Point1 0.840486 0.000000 0.744968
; Point2 0.549500 0.000000 1.181163
;; t10
```

pt–cvty–info:angle

Scheme Extension: Model Geometry

Action: Returns the angle recorded in a pt-cvty-info.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**pt–cvty–info:angle** pt–cvty–info)

Arg Types: pt–cvty–info scm_pt_cvty_info

Returns: real

Errors: None

Description: Returns the angle stored in a pt-cvty-info for the point along an edge for which the pt-cvty-info was recorded. Negative numbers infer concave, positive numbers infer convex.

pt–cvty–info represents the convexity of a single point along the edge.

Limitations: None

Example:
```
; pt–cvty–info:angle
; Create a block
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(pt–cvty–info:angle (edge:mid-pt–cvty-info
    (list–ref edge–list 0)))
;; 1
```

pt–cvty–info:instantiate

Scheme Extension: Model Geometry, Model Topology

Action: Returns the convexity of a pt-cvty-info for a given smooth angle tolerance.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**pt–cvty–info:instantiate** pt–cvty–info tol)

Arg Types:

pt–cvty–info	scm_pt_cvty_info
tol	real

Returns: scm_pt_cvty_info

Errors: None

Description: Returns the convexity of the given pt-cvty-info, for the given tolerance that determines when an edge is regarded as ”smooth”. The tolerance may be passed as –1 to emulate the historical behavior of edge:convexity. Equivalent to the C++ pt_cvty_info::instantiate.

pt–cvty–info represents the convexity of a single point along the edge.

Limitations: None

Example:

```
; pt–cvty–info:instantiate
; Defines some geometry and return the convexity.
(define w (solid:wiggle 60 60 60 ”sym”))
;; w
(define edge (list–ref (entity:edges w) 0))
;; edge
(pt–cvty–info:instantiate
    (edge:mid–pt–cvty–info edge) 0.01)
;; #[cvty: cvx]
```

pt–cvty–info:tangent–convexity

Scheme Extension: Model Geometry, Model Topology

Action: Returns the tangent convexity of a pt-cvty-info for a given smooth angle tolerance.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**pt–cvty–info:tangent–convexity** pt–cvty–info)

Arg Types: pt–cvty–info — scm_pt_cvty_info

Returns: scm_pt_cvty_info

Errors: None

Description: Returns the convexity of the given pt-cvty-info, for the given tolerance that determines when an edge is regarded as “smooth.” Equivalent to the C++ pt_cvty_info::tangent_convexity.

pt–cvty–info represents the convexity of a single point along the edge.

Limitations: None

Example:

```
; pt–cvty–info:tangent–convexity
; Create a block with a wiggle top
(define w (solid:wiggle 60 60 60 ”sym”))
;; w
; Define the edges of the block
(define edge (list–ref (entity:edges w) 0))
;; edge
(pt–cvty–info:tangent–convexity
    (edge:mid–pt–cvty–info edge))
;; #[cvty: cvx knf]
```

pt–cvty–info:unknown

Scheme Extension: Model Geometry, Model Topology

Action: Determines whether a pt-cvty-info is unknown.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**pt–cvty–info:unknown** pt–cvty–info)

Arg Types: pt–cvty–info scm_pt_cvty_info

Returns: boolean

Errors: None

Description: Returns #t if the given pt-cvty-info is unknown; e.g., it could not be computed for some reason.

pt–cvty–info represents the convexity of a single point along the edge.

Limitations: None

Example:

```
; pt–cvty–info:unknown
; Create a block
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(pt–cvty–info:unknown (edge:mid–pt–cvty–info
    (list–ref edge–list 0)))
;; #f
```

pt–cvty–info:unset

Scheme Extension: Model Geometry, Model Topology

Action: Returns whether a pt-cvty-info is unset.

Filename: intr/intr_scm/cvty_typ.cxx

APIs: None

Syntax: (**pt–cvty–info:unset** pt–cvty–info)

Arg Types:	pt–cvty–info	scm_pt_cvty_info

Returns: boolean

Errors: None

Description: Returns #t if a pt-cvty-info is unset. This should never be the case, but is provided for completeness.

pt–cvty–info represents the convexity of a single point along the edge.

Limitations: None

Example:

```
; pt–cvty–info:unset
; Create a block
(define block1 (solid:block (position 0 10 0)
    (position 10 20 20)))
;; block1
; Define the edges of block1
(define edge–list (entity:edges block1))
;; edge–list
(pt–cvty–info:unset (edge:mid–pt–cvty–info
    (list–ref edge–list 0)))
;; #f
```

solid:classify–position

Scheme Extension: Object Relationships

Action: Classifies a point with respect to a solid.

Filename: intr/intr_scm/rtst_scm.cxx

APIs: api_point_in_body

Syntax:

```
(solid:classify–position entity test–position
     [use–boxes] [acis–opts])
```

Arg Types:	entity	entity
	test–position	position
	use–boxes	boolean
	acis–opts	acis–options

Returns: integer

Errors: First object is not a solid.
Second object is not a position.

Description: This extension returns –1 if the position is inside the solid, 0 if the position is on the surface of the solid, or 1 for the position is outside the solid. This extension returns #f if the containment cannot be determined.To test when the body may be a void, set the use–boxes flag to FALSE. This will make testing slower, however.

The argument entity specifies a solid body entity.

The argument test–position specifies the location on the entity to classify.

The optional argument use–boxes, when set to #f, tests when the body may be void.

The optional argument acis–opts helps enable journaling and versioning options.

Limitations: None

Example:

```
; solid:classify-position
; Create a solid block.
(define block1 (solid:block (position 0 0 0)
    (position 15 15 15)))
;; block1
; Classify point 1 with respect to the solid.
(solid:classify-position block1 (position 6 3 8))
;; -1
(solid:classify-position block1 (position 0 0 0))
;; 0
; Classify point 2 with respect to the solid.
(solid:classify-position block1
    (position -4 -4 -4))
;; 1
```

solid:ray–test

Scheme Extension: Object Relationships

Action: Gets the position where a ray intersects a solid.

Filename: intr/intr_scm/rtst_scm.cxx

APIs: api_ray_test_body

Syntax: (**solid:ray-test** entity ray [acis-opts])

Arg Types:		
	entity	body
	ray	ray
	acis-opts	acis–options

Returns: ((entity . position) ...)

Errors: None

Description: Returns the position(s) where a ray intersects a solid.

The argument entity must be a solid body.

The argument ray consists of a position and a direction.

The optional argument acis–opts helps enable journaling and versioning options.

This extension returns pairs. The first element of each pair is the entity hit by the ray, and the second element of the pair is the position where the ray intersects the solid. The pairs are sorted along the direction of the ray. If the ray intersects a single face more than once, the extension returns the first intersection.

Limitations: None

Example:

```
; solid:ray–test
; Create a solid block.
(define block1 (solid:block (position 0 0 0)
    (position 40 40 40)))
;; block1
; Determine where the ray intersects the solid block.
(solid:ray–test block1 (ray (position 0 0 0)
    (gvector 0 0 1)))
;; ((#[entity 4 1] . #[position 0 0 0])
;; (#[entity 3 1] . #[position 0 0 40]))
```

tcoedge–bad–crv:csl

Scheme Extension: Scheme Interface

Action: Returns information from a tm–chk–info of derived type tcoedge–bad–crv.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tcoedge–bad–crv:csl** tm–check)

Arg Types: tm–check tm–chk–info

Returns: string

Errors: Argument is not a tcoedge_bad_crv.

Description: Returns the information from the check_status enum (see kernel/kernint/d3_chk/chk_stat.hxx). For example, this function could return (”check_non_C1” ”check_self_intersects”).

tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tcoedge–bad–crv:csl
; Requires a bad sat file to generate this error.
; Example/bad sat file not available at this time.
```

tcoedge–bad–crv?

Scheme Extension: Scheme Interface

Action: Returns #t when the given object is a tm–chk–info of derived type tcoedge_bad_crv.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tcoedge–bad–crv?** tm–check)

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tcoedge–bad–crv?
; Build block for example.
(define block (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block
(define errors (tm–check:all (car
    (entity:edges block))))
;; errors
(tcoedge–bad–crv? (car errors))
;; #f
```

tcoedge–bs2–non–g1?

Scheme Extension: Scheme Interface

Action: Return #t if the given object is a tm-chk-info of derived type tcoedge_bs2_non_G1.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tcoedge–bs2–non–g1?** t)

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:

```
; tcoedge–bs2–non–g1
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tcoedge–bs2–non–g1? (car errors))
;; #f
```

tcoedge–bs2–outside–sf?

Scheme Extension: Scheme Interface

Action: Return #t if the given object is a tm-chk-info of derived type tcoedge_bs2_outside_sf.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tcoedge–bs2–outside–sf?** tm–check)

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tcoedge–bs2–outside–sf?
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tcoedge–bs2–outside–sf? (car errors))
;; #f
```

tcoedge–crv–non–g1?

Scheme Extension: Scheme Interface

Action: Return #t if the given object is a tm-chk-info of derived type tcoedge_crv_non_G1.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tcoedge–crv–non–g1?** tm–check)

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tcoedge–crv–non–g1?
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tcoedge–crv–non–g1? (car errors))
;; #f
```

tedge–bad–crv:csl

Scheme Extension: Scheme Interface

Action: Return the check_status_list from a tm_chk_info of derived type tedge_bad_crv.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: `(tedge–bad–crv:csl tm–check)`

Arg Types: tm–check tm–chk–info

Returns: (string...)

Errors: Argument is not a tedge_bad_crv.

Description: A list of strings is returned, being the printed representation of the check_status enum (see kernel/kernint/d3_chk/chk_stat.hxx). For example, this extension could return (”check_non_C1” ”check_self_intersects”).

tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:

```
; tedge–bad–crv:csl
; Requires a bad sat file to generate this error.
; Example/bad sat file not available at this time.
```

tedge–bad–crv?

Scheme Extension: Scheme Interface

Action: Return #t if the given object is a tm-chk-info of derived type tedge_bad_crv.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: `(tedge–bad–crv? tm–check)`

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tedge–bad–crv?
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tedge–bad–crv? (car errors))
;; #f
```

tedge–crv–non–g1?

Scheme Extension: Scheme Interface

Action: Return #t if the given object is a tm-chk-info of derived type tedge_crv_non_G1.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tedge–crv–non–g1?** tm–check)

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tedge–crv–non–g1?
; Create geometry to test command.
(define b (solid:block
    (position 0 0 0) (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tedge–crv–non–g1? (car errors))
;; #f
```

tedge–local–self–int?

Scheme Extension: Scheme Interface

Action: Return #t if the given object is a tm-chk-info of derived type tedge_local_self_int.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tedge–local–self–int?** tm–check)

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:

```
; tedge–local–self–int?
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tedge–local–self–int? (car errors))
;; #f
```

tedge–remote–self–int:other–edge–param

Scheme Extension: Scheme Interface

Action: Returns the other parameter value of the self–intersection.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tedge–remote–self–int:other–edge–param** tm–check)

Arg Types: tm–check tm–chk–info

Returns: real | boolean

Errors: None

Description: The self–intersection is represented by two parameter values. The Scheme extension tm–chk–info:edge–param returns the first; this Scheme extension returns the second (other) parameter value of the self–intersection.

tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:

```
; tedge–remote–self–int:other–edge–param
; Create geometry to illustrate command.
(define block (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block
; Get list of all edges on block.
(define edges (car (entity:edges block)))
;; edges
(tm–check:all edges)
;; (#[tm_bad_topology 58e23b8])
(tedge–remote–self–int? edges)
;; #f
(define t (tedge–remote–self–int:other–edge–param
    (car edges))
;; t
```

tedge–remote–self–int?

Scheme Extension: Scheme Interface

Action: Returns #t if the given object is a tm-chk-info of derived type tedge_remote_self_int.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tedge–remote–self–int?** tm–check)

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tedge–remote–self–int?
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tedge–remote–self–int? (car errors))
;; #f
```

tedge–tcoedge–bad–geom?

Scheme Extension: Scheme Interface

Action: Returns #t if the given object is a tm-chk-info of derived type tedge_tcoedge_bad_geom.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tedge–tcoedge–bad–geom?** tm–check)

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tedge–tcoedge–bad–geom?
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tedge–tcoedge–bad–geom? (car errors))
;; #f
```

tedge–tcoedge–bad–tol?

Scheme Extension: Scheme Interface

Action: Returns #t if the given object is a tm-chk-info of derived type tedge_remote_self_int.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tedge–tcoedge–bad–tol** tm–check)

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:

```
; tedge–tcoedge–bad–tol?
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tedge–tcoedge–bad–tol? (car errors))
;; #f
```

tedge–tcoedge–ranges:start

Scheme Extension: Scheme Interface

Action: Return the start flag from a tm_chk_info of derived type tedge_tcoedge_ranges.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tedge–tcoedge–ranges:start** tm–check)

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: Argument is not a tedge_tcoedge_ranges.

Description: The start flag from a tedge-tcoedge-ranges error object is returned.

tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tedge–tcoedge–ranges:start
; Requires a bad sat file to generate this error.
; Example/bad sat file not available at this time.
```

tedge–tcoedge–ranges?

Scheme Extension: Scheme Interface

Action: Returns #t if the given object is a tm-chk-info of derived type tedge_tcoedge_ranges.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tedge–tcoedge–ranges?** tm–check)

Arg Types: tm–check tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tedge–tcoedge–ranges?
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tedge–tcoedge–ranges? (car errors))
;; #f
```

tm–bad–topology?

Scheme Extension: Scheme Interface

Action: Returns #t if the given object is a tm-chk-info of derived type tm_bad_topology.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tm–bad–topology?** tm–check)

Arg Types:	tm–check	tm–chk–info

Returns: boolean

Errors: None

Description: tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:

```
; tm–bad–topology?
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tm–bad–topology? (car errors))
;; #t
```

tm–check:all

Scheme Extension: Scheme Interface

Action: Performs all the tolerant modelling geometry checks on this edge and its coedges.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:all** edge)

Arg Types:	edge	edge

Returns: (edge...)

Errors: Argument not an EDGE.

Description: Performs all of the tolerant modeling geometry tests on this tedge and its coedges. Therefore it invokes: tm-check:tm-bad-topology, tm-check:tedge, tm-check:tcoedge (for each coedge), tm-check:tedge-tcoedge (for each coedge), tm-check:tedge-self-int, and tm-check:tedge-tol. A list of any tm-chk-info error objects that are generated is returned (or an empty list if all is fine). Note that in general, if an early check fails, then the later checks may be omitted (on the grounds that they are not even guaranteed to work unless certain preconditions are guaranteed.)

edge is an argument of type edge on which tolerant modelling geometry checks are performed.

Limitations: None

Example:
```
; tm–check:all
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix
    (car (entity:edges b)))))
;; e
(tm–check:all e)
;; ()
; this tedge is fine
```

tm–check:tcoedge

Scheme Extension: Scheme Interface

Action: Performs all the tolerant edge tests that apply to the parameter and 3–space curves of this coedge. (Amalgamates a number of basic tests.).

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tcoedge** coedge)

Arg Types: coedge coedge

Returns: (coedge...)

Errors: Argument not a TCOEDGE.

Description: Performs all the tm-check tests that apply to this coedge's parameter–space and 3–space curves, without reference to the edge itself. This test performs in order: tm-check:tcoedge-b2-non-g1, tm-check:tcoedge-bs2-outside-sf, tm-check:tcoedge-crv-non-g1, tm-check:tcoedge-bad-crv. A list of any tm-chk-info error objects that are generated is returned (or an empty list if all is fine).

coedge is an argument of type tcoedge on which tolerant edge tests are to be performed.

Limitations: None

Example:
```
; tm–check:tcoedge
; Create geometry to illustrate command.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix
    (car (entity:edges b)))))
;; e
(tm–check:tcoedge (car (entity:tcoedges e)))
;; ()
; coedge geometry is fine
```

tm–check:tcoedge–bad–crv

Scheme Extension: Scheme Interface

Action: Tests validity of the 3D geometry of a tolerant coedge.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tcoedge–bad–crv** coedge)

Arg Types: coedge coedge

Returns: (coedge...)

Errors: argument not a TCOEDGE.

Description: Returns as tm-check:tm-bad-topology would if the coedge's edge is not a TEDGE. Otherwise it checks the coedge 3D curve geometry for being invalid, according to the standard ACIS curve checker. If any problems are identified, a tm-chk-info of derived type tcoedge-bad-crv is returned which lists (via tcoedge-bad-crv:csl) the check_status_list (see kernel/kernint/d3_chk/chk_stat.hxx) of problems returned by the curve checker. If the geometry is valid, an empty list is returned.

coedge is an argument of type tcoedge whose 3D geometry is to be tested.

Limitations: None

Example:
```
; tm–check:tcoedge–bad–crv
; Create geometry to illustrate command.
(define b (solid:block
    (position 0 0 0) (position 10 10 10)))
;; b
(define e (car (tolerant:fix
    (car (entity:edges b)))))
;; e
(tm–check:tcoedge–bad–crv (car (entity:tcoedges e)))
;; ()
; geometry is valid
```

tm–check:tcoedge–bs2–non–g1

Scheme Extension: Scheme Interface

Action: Tests whether the parameter curve (bs2_curve) of a tcoedge is G1 in parameter space.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tcoedge–bs2–non–g1** coedge)

Arg Types: coedge tcoedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TCOEDGE.

Description: Returns as tm–check:tm–bad–topology would if the coedge's edge is not a TEDGE. Otherwise it checks the coedge parameter space curve for being G1 in parameter space. If any problems are identified, a tm–chk–info of derived type tcoedge–bs2–non–g1 is returned which lists (via tm–chk–info:coedge–param) the coedge parameter where there is a problem. Note that the coedge parameter is negated if the coedge is reversed to match the edge's sense. If the parameter curve is G1, an empty list is returned.

coedge is an argument of type tcoedge whose parametric curve is tested for the above mentioned condition.

Limitations: None

Example:
```
; tm–check:tcoedge–bs2–non–g1
; Check to see if the parameter curve is G1 or not.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tcoedge–bs2–non–g1
    (car (entity:tcoedges e)))
;; ()
; bs2_curve is G1
```

tm–check:tcoedge–bs2–outside–sf

Scheme Extension: Tolerant Modeling

Action: Tests whether the parameter curve (bs2_curve) of a tcoedge strays outside the boundary of the coedge's face's surface.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tcoedge–bs2–outside–sf** coedge)

Arg Types: coedge tcoedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TEDGE.

Description: Returns as tm–check:tm–bad–topology would if coedge's edge is not a TEDGE. Otherwise, checks the coedge parameter space curve to see if it strays outside the boundaries of the coedge's face's surface. The resolution used is resabs in parameter and will pick up most significant problems. If any problems are identified, a tm–chk–info of derived type tcoedge–bs2–outside–sf is returned which lists (via tm–chk–info:coedge–param) the coedge parameter where there is a problem. The coedge parameter is negated if the coedge is reversed so as to have the same sense as the edge itself. If the parameter curve is fine in this respect, an empty list is returned.

coedge is an argument of type tcoedge whose parametric curve is tested for the above mentioned condition.

Limitations: None

Example:
```
; tm–check:tcoedge–bs2–outside–sf
; Check a parameter curve of a coedge.
(define block (solid:block (position 0 0 0)
    (position 10 10 10)))
;; block
(define list (car (tolerant:fix (car
    (entity:edges block)))))
;; list
(tm–check:tcoedge–bs2–outside–sf (car
    (entity:tcoedges block)))
;; ()
; bs2_curve lies within surface
```

tm–check:tcoedge–crv–non–g1

Scheme Extension: Tolerant Modeling

Action: Test whether the 3D curve of a tolerant coedge is G1 or not.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tcoedge–crv–non–g1** edge)

Arg Types: coedge tcoedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TEDGE.

Description: Returns as tm–check:tm–bad–topology would if the coedge is not a TCOEDGE. Otherwise it checks the coedge 3D curve for being G1. If any problems are identified, a tm–chk–info of derived type tcoedge–crv–non–g1 is returned which lists (via tm–chk–info:coedge–param) the coedge parameter where there is a problem. Note that the coedge parameter is negated if the coedge is reversed so as to have the same sense as the edge itself. If the 3D curve is G1, an empty list is returned.

edge is an argument of type TEDGE whose 3D cure is tested for G1 continuity.

Limitations: None

Example:

```
; tm–check:tcoedge–crv–non–g1
; Check to see if the coedge is G1 or not.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tcoedge–crv–non–g1 (car
    (entity:tcoedges e)))
;; ()
; 3D curve is G1
```

tm–check:tedge

Scheme Extension: Tolerant Modeling

Action: Performs all the tolerant edge tests that apply to the curve of this edge. (Amalgamates together a number of more basic tests.).

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tedge** edge)

Arg Types: edge tedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TEDGE.

Description: Performs all the tm–check tests that apply to this edge curve, without reference to any of the edge's coedges. This test performs in order: tm–check:tedge–crv–non–g1, tm–check:tedge–bad–crv. A list of any tm–chk–info error objects that are generated is returned (or an empty list if all is fine).

edge is an argument of type TEDGE on which all the tolerant edge tests are performed.

Limitations: None

Example:

```
; tm–check:tedge
; Check all tolerant edge checks.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tedge e)
;; ()
; edge geometry is fine
```

tm–check:tedge–bad–crv

Scheme Extension: Tolerant Modeling

Action: Test whether the geometry of a tolerant edge is illegal or not.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tedge–bad–crv** edge)

Arg Types: edge tedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TEDGE.

Description: Returns as tm–check:tm–bad–topology would if the edge is not a TEDGE. Otherwise it checks whether the edge geometry is invalid, according to the standard ACIS curve checker. If any problems are identified, a tm–chk–info of derived type tedge–bad–crv is returned which lists (via tedge–bad–crv:csl) the check_status_list of problems returned by the curve checker. If the geometry is valid, an empty list is returned.

edge is an argument of type TEDGE for which the geometry is tested for legality.

Limitations: None

Example:

```
; tm–check:tedge–bad–crv
; Check to see if the edge is a legal edge.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tedge–bad–crv e)
;; ()
; geometry is valid
```

tm–check:tedge–crv–non–g1

Scheme Extension: Tolerant Modeling

Action: Test whether the geometry of a tolerant edge is G1 or not.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tedge–crv–non–g1** edge)

Arg Types: edge tedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TEDGE.

Description: Returns as tm–check:tm–bad–topology would if the edge is not a TEDGE. Otherwise it checks the edge geometry for being G1. If it is not, a list containing a tm–chk–info of derived type tedge_crv_non_g1 is returned (and tm–chk–info:edge–param gives the parameter of the problem). If the geometry is G1, an empty list is returned.

edge is an argument of type TEDGE for which the geometry is tested for G1 continuity.

Limitations: None

Example:

```
; tm–check:tedge–crv–non–g1
; Check to see if the curve is G1 or not.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tedge–crv–non–g1 e)
;; ()
; geometry is valid
```

tm–check:tedge–local–self–int

Scheme Extension: Tolerant Modeling

Action: Tests whether the geometry of a tolerant edge has local self–intersections (i.e., creases up on itself).

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tedge–local–self–int** edge)

Arg Types: edge tedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TEDGE.

Description: Returns as tm–check:tm–bad–topology would if the edge is not a TEDGE. Otherwise it checks the edge geometry for having local self–intersections (i.e., regions where the tube around the edge, of radius equal to the edge tolerance, creases up on itself). If so, a list containing a tm–chk–info of derived type tedge–local–self–int is returned. This may indicate the edge parameter of the problem (via tm–chk–info:edge–param). If there are no local self–intersections, an empty list is returned.

edge is an argument of type TEDGE for which the local self–intersections are tested.

Limitations: None

Example:

```
; tm–check:tedge–local–self–int
; Check for edge local self–intersections.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tedge–local–self–int e)
;; ()
; geometry is valid
```

tm–check:tedge–remote–self–int

Scheme Extension: Tolerant Modeling

Action: Tests the geometry of a tolerant edge to determine remote self–intersections.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tedge–remote–self–int** edge)

Arg Types: edge tedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TEDGE.

Description: Returns as tm–check:tm–bad–topology would if the edge is not a TEDGE. Otherwise it checks the edge geometry for having remote self–intersections (i.e., regions where the tube around the edge, of radius equal to the edge tolerance, returns to re–intersect itself). If so, a list containing a tm–chk–info of derived type tedge–remote–self–int is returned. This may indicate the two edge parameters of the problem (via tm–chk–info:edge–param and tedge–remote–self–int:other–edge–param). If there are no remote self–intersections, an empty list is returned.

edge is an argument of type TEDGE for which the remote self–intersections are tested.

Limitations: None

Example:

```
; tm–check:tedge–remote–self–int
; Check for edge remote self–intersections.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tedge–remote–self–int e)
;; ()
; geometry is valid
```

tm–check:tedge–self–int

Scheme Extension: Tolerant Modeling

Action: Tests a tolerant edge for local and remote self–intersections. (Amalgamates a number of more basic tests).

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tedge–self–int** edge)

Arg Types: edge tedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TEDGE.

Description: Tests whether this TEDGE has any local or remote self–intersections. This tests uses the edge tolerance, so the checks tm–check:tedge and then tm–check:tcoedge and tm–check:tedge–tcoedge (for each coedge of the edge) should already have been passed. This test then performs in order: tm–check:tedge–local–self–int, tm–check:tedge–remote–self–int. A list of any tm–chk–info error objects that are generated is returned (or an empty list if all is fine).

edge is an argument of type TEDGE for which the local and remote self–intersections are tested.

Limitations: None

Example:

```
; tm–check:tedge–self–int
; Check for edge self–intersection.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tedge–self–int e)
;; ()
; edge geometry does not self–intersect
```

tm–check:tedge–tcoedge

Scheme Extension: Tolerant Modeling

Action: Performs all the tolerant edge tests that ensure that this edge geometry and coedge geometry are compatible for tolerant modeling.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tedge–tcoedge** coedge)

Arg Types: coedge tcoedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TCOEDGE.

Description: Performs all the tm–check tests that apply to this coedge and edge together, to ensure that they are compatible. It therefore performs in order: tm–check:tedge–tcoedge–ranges, tm–check:tedge–tcoedge–bad–geom. (It is assumed that the edge and coedge have already been individually checked for validity using tm–check:tedge and tm–check:tcoedge.) A list of any tm–chk–info error objects that are generated is return (or an empty list if all is fine).

coedge is an argument of type TCOEDGE for which the compatibility is tested.

Limitations: None

Example:

```
; tm–check:tedge–tcoedge
; Check all tests for edge tolerance.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tedge–tcoedge (car (entity:tcoedges e)))
;; ()
; coedge geometry is fine
```

tm–check:tedge–tcoedge–bad–geom

Scheme Extension: Tolerant Modeling

Action: Test whether the 3D curves of a tcoedge and its edge have compatible directions and curvatures all along their length.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tedge–tcoedge–bad–geom** coedge)

Arg Types: coedge tcoedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TCOEDGE.

Description: Returns as tm–check:tm–bad–topology would if the coedge's edge is not a TEDGE. Otherwise it analyses the coedge 3D curve and the curve of its edge to determine whether they have sympathetic directions and curvatures all along their length. This is necessary for tolerant modeling to work properly.

coedge is an argument of type TCOEDGE for which the compatibility is tested.

Limitations: None

Example:

```
; tm–check:tedge–tcoedge–bad–geom
; Check the coedge for bad geometry.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tedge–tcoedge–bad–geom (car
    (entity:tcoedges e)))
;; ()
; geometry is valid
```

tm–check:tedge–tcoedge–bad–tol

Scheme Extension: Tolerant Modeling

Action: Test whether the 3D curve of a tcoedge strays outside the computed tolerance zone of a tolerant edge.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tedge–tcoedge–bad–tol** coedge)

Arg Types: coedge tcoedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TCOEDGE.

Description: Returns as tm–check:tm–bad–topology would if the coedge's edge is not a TEDGE. Otherwise it analyses the coedge 3D curve and the curve of its edge to determine whether the coedge strays further from the edge than the edge's tolerance.

Limitations: None

Example:

```
; tm–check:tedge–tcoedge–bad–tol
; Check the coedge for bad tolerance.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tedge–tcoedge–bad–tol (car
    (entity:tcoedges e)))
;; ()
; geometry is valid
```

tm–check:tedge–tcoedge–ranges

Scheme Extension: Tolerant Modeling

Action: Test whether the ends of the 3D curves of a tcoedge and its edge are sufficiently compatible for tolerant modeling to work.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tedge–tcoedge–ranges** coedge)

Arg Types: coedge tcoedge

Returns: (tm–chk–info ...)

Errors: Argument is not a TCOEDGE.

Description: Returns as tm–check:tm–bad–topology would if the coedge's edge is not a TEDGE. Otherwise it analyses the start and end of the coedge 3D curve and the curve of its edge to determine whether the curves overlap sensibly and have compatible directions so that tolerant modeling can work on them.

coedge is an argument of type TCOEDGE for which the compatibility is tested.

Limitations: None

Example:

```
; tm–check:tedge–tcoedge–ranges
; Check the coedge and edge for compatibility.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm–check:tedge–tcoedge–ranges (car
    (entity:tcoedges e)))
;; ()
; geometry is valid
```

tm–check:tedge–tol

Scheme Extension: Tolerant Modeling

Action: Test whether the recorded tolerance of this TEDGE is correct.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tedge–tol** edge)

Arg Types: tedge tedge

Returns: (tm–chk–info ...)

Errors: Argument not an TEDGE.

Description: Tests whether the recorded tolerance of this tedge is correct. As this test uses the edge tolerance, the checks tm–check:tedge and then tm–check:tcoedge and tm–check:tedge–tcoedge (for each coedge of the edge) should already have been passed. This test then performs in order: tm–check:tedge–tcoedge–bad–tol for each coedge of the edge. A list of any tm–chk–info error objects that are generated is returned (or an empty list if all is fine).

tedge is an argument of type tedge for which the correctness of recorded tolerance is tested.

Limitations: None

Example:

```
; tm-check:tedge-tol
; Check the recorded tolerance of a tedge.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define e (car (tolerant:fix (car
    (entity:edges b)))))
;; e
(tm-check:tedge-tol e)
;; ()
; tolerance is correct
```

tm–check:tm–bad–topology

Scheme Extension: Tolerant Modeling

Action: Test whether the topology of a given edge is correct for it being a tolerant edge.

Filename: intr/intr_scm/tmchk_scm.cxx

APIs: None

Syntax: (**tm–check:tm–bad–topology** edge)

Arg Types: tedge tedge

Returns: (tm–chk–info ...)

Errors: Argument not an EDGE.

Description: Checks whether the given EDGE is actually a TEDGE; that it has geometry; that it has at least one COEDGE; that all its COEDGEs are actually TCOEDGEs, and that all have geometry. If any of these tests fails, returns a list of one tm–chk–info of derived type tm–bad–topology (otherwise an empty list if all is fine). This error object points to the edge that was checked; if the problem actually involved one of the coedges then that coedge is pointed to also.

tedge is an argument of type tedge for which the correctness of topology is tested.

Limitations: None

Example:

```
; tm–check:tm–bad–topology
; Check for bad topology and return the information.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:tm–bad–topology
    (car (entity:edges b))))
;; errors
; the EDGE was not a TEDGE
```

tm–chk–info:coedge

Scheme Extension: Tolerant Modeling

Action: Return the coedge that caused a particular error.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tm–chk–info:coedge** tm–check)

Arg Types: tm–check tm–chk–info

Returns: entity | boolean

Errors: None

Description: Returns the coedge recorded within the tm–chk–info that was the cause of the reported error, or #f if no edge is recorded.

tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:

```
; tm–chk–info:coedge
; Print out the coedge where an error occurs.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tm–chk–info:coedge (car errors))
;; #f
```

tm–chk–info:coedge–param

Scheme Extension: Tolerant Modeling

Action: Return the coedge parameter, if known, where a particular error occurs. The coedge parameter is corrected to have the same sense as the edge.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: `(tm–chk–info:coedge–param tm–check)`

Arg Types: tm–check tm–chk–info

Returns: real | boolean

Errors: None

Description: Returns the coedge parameter recorded within the tm–chk–info where the particular error being reported occurs, or #f if no coedge parameter is present.

tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:

```
; tm–chk–info:coedge–param
; Print out the coedge parameter of a given tolerant
; model information set.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tm–chk–info:coedge–param (car errors))
;; #f
```

tm–chk–info:edge

Scheme Extension: Tolerant Modeling

Action: Return the edge that caused a particular error.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: `(tm–chk–info:edge tm–check)`

Arg Types: tm–check tm–chk–info

Returns: real | boolean

Errors: None

Description: Returns the edge recorded within the tm–chk–info that was the cause of the reported error, or #f if no edge is recorded.

tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tm–chk–info:edge
; Print out the edge where an error occurs.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(define checks (tm–chk–info:edge (car errors)))
;; checks
```

tm–chk–info:edge–param

Scheme Extension: Tolerant Modeling

Action: Return the edge parameter, if known, where a particular error occurs.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: `(tm–chk–info:edge–param tm–check)`

Arg Types: tm–check tm–chk–info

Returns: real | boolean

Errors: None

Description: Returns the edge parameter recorded within the tm–chk–info where the particular error being reported occurs, or #f if no edge parameter is present.

tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

Limitations: None

Example:
```
; tm–chk–info:edge–param
; Print out the edge parameter of a given tolerant
; model information set.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tm–chk–info:edge–param (car errors))
;; #f
```

tm–chk–info:print

Scheme Extension: Tolerant Modeling

Action: Print a full and relatively readable description of the entire contents of a tm–chk–info on an output port, terminating with a newline.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tm–chk–info:print** tm–check [port])

Arg Types:		
	tm–check	tm–chk–info
	port	string

Returns: tm–chk–info

Errors: None

Description: Prints a full and relatively readable description of the entire contents of the tm–chk–info (first argument) on the given output port (second argument), defaulting to the current output port if the second argument is omitted. The output terminates with a newline.

tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

port specifies the output port to which the the contents of tm–check are written and the output is terminated with a newline.

Limitations: None

Example:
```
; tm–chk–info:print
; Print out the contents of a given tolerant model
; information set.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tm–chk–info:print (car errors))
; tm_bad_topology: edge 20469008
;; #[tm_bad_topology 4059b210]
```

tm–chk–info:print1

Scheme Extension: Tolerant Modeling

Action: Print a full and relatively readable description of the entire contents of a tm–chk–info on an output port, terminating without a newline.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tm–chk–info:print1** tm–check [port])

Arg Types:

tm–check	tm–chk–info
port	string

Returns: tm–chk–info

Errors: None

Description: Prints a full and relatively readable description of the entire contents of the tm–chk–info (first argument) on the given output port (second argument), default to the current output port if the second argument is omitted. The output is terminated without a newline.

tm–check is an object of type tm–chk–info that contains tolerant modeling check details.

port specifies the output port to which the the contents of tm–check are written and the output is terminated without a newline.

Limitations: None

Example:

```
; tm–chk–info:print1
; Print out the contents of a given tolerant model
; information set.
(define b (solid:block (position 0 0 0)
    (position 10 10 10)))
;; b
(define errors (tm–check:all (car (entity:edges b))))
;; errors
(tm–chk–info:print1 (car errors))
; tm_bad_topology: edge 1413712
;; #[tm_bad_topology 4059b210]
```

tm–chk–info?

Scheme Extension: Tolerant Modeling

Action: Determines whether the given Scheme object is a tm–chk–info.

Filename: intr/intr_scm/tmchk_typ.cxx

APIs: None

Syntax: (**tm–chk–info?** object)

Arg Types:

object	scheme–object

Returns: boolean

Errors: None

Description: Returns #t if the given object is a tm–chk–info.

object is a variable of type scheme–object. The scheme tests if this object is a tm–chk–info.

Limitations: None

Example:

```
; tm–chk–info?
; Check to see if a Scheme Object is a
; tolerant model.
(define cube (solid:block (position 0 0 0)
    (position 10 10 10)))
;; cube
(define errors (tm–check:all (car
    (entity:edges cube))))
;; errors
(tm–chk–info? (car errors))
;; #t
```

wire–body:self–intersect?

Scheme Extension: Intersectors

Action: Checks to see if a wire-body intersects itself.

Filename: intr/intr_scm/icrv_scm.cxx

APIs: None

Syntax: `(wire–body:self–intersect? wire–body [acis–opts])`

Arg Types:

wire–body	wire–body
acis–opts	acis–options

Returns: boolean

Errors: None

Description: This extension tests to see if a wire-body self intersects. If a self intersection is found, the extension returns true. Otherwise it returns false.

wire–body is an input variable of type wire–body that needs to be tested for self intersection.

acis–opts is an acis–options variable. This is used to enable versioning and journaling options.

Limitations: None

Example:

```
; wire–body:self–intersect?
; No example available at this time.
```