Chapter 41.
Law Symbols

Topic: Ignore

In ACIS, *law mathematical functions* (laws) can be used to define geometry and solve mathematical problems. A law is a character string made up of valid *law symbols* enclosed within quotation marks. The law symbols used in law functions are very similar to common mathematical notation and to the adaptation of mathematical notation for use in computers. Refer to the *3D ACIS Online Help User's Guide* for a description of the fields in the reference template.

DRAFT

Law Symbol: Laws

Action: Creates a law for accepting drafts as input data.

Derivation: draft_law : multiple_law : law : ACIS_OBJECT : –

Syntax: **DRAFT1**

Description: Refer to Action.

Example:
```
; law "DRAFT"
; Create a simple law function.
```

EDGE

Law Symbol: Laws

Action: Creates a law for accepting edges as input data.

Derivation: curve_law_data : base_curve_law_data : path_law_data : law_data : ACIS_OBJECT : –

Syntax: **EDGE1**

Description: Refer to Action.

Example:
```
; law "EDGE"
; Create a simple law function.
```

PCURVE

Law Symbol: Laws

Action: Creates a law for accepting pcurves as input data.

Derivation: pcurve_law_data : base_pcurve_law_data : path_law_data : law_data : ACIS_OBJECT : –

Syntax: **PCURVE**

Description: Refer to Action.

Example:
```
; law "PCURVE"
; Create a simple law function.
```

SURF

Law Symbol: Laws

Action: Creates a law that returns the positions of the defining surface.

Derivation: surface_law_data : base_surface_law_data : law_data : ACIS_OBJECT : –

Syntax: **SURF**

Description: surf returns the positions of the defining surface at the parameter value. This law symbol is a way to pass a surface into a law for other purposes, such as evaluation. The dimension of the input, my_surface_law_data, is two, but when surf is evaluated, it returns an item in three dimensions.

ACIS defines its own parameter range for a surface which is used by this law.

Example:

```
; law ”SURF”
; Create a surface to evaluate. (define my_sphere
(solid:sphere (position 0 0 0) 10))
;; my_sphere
; => #[entity 2 1]
(define my_surflaw (law ”surf(surf1)”
(car (entity:faces my_sphere))))
;; my_surflaw
; => #[law ”SURF(SURF1)”]
(define my_sveclaw (law ”surfvec(law1,
vec(x,y,z), vec(a4, a5))” my_surflaw))
;; my_sveclaw
; my_sveclaw =>
; [SURFVEC(SURF(SURF1),VEC(X,Y,Z), VEC(A4,A5))” ]
(law:eval my_sveclaw (list 0 0 1 0 0))
;; (1 0 0 0)

; The law created takes an xyz vector and a uv
; position on the surface. It returns a uv vector
; in the direction of the given xyz vector at the
; given uv position on the surface. It also returns
; as the last two arguments the uv positions. The uv
; position is echoed.
; Here is an example at the pole.
(law:eval my_sveclaw
(list 1 1 0 (law:eval ”pi/2”) 0))
;; (–1 0 1.5707963267949 0.785398163401155)
; At the pole, this response means that you have to
; turn v by pi/4 to get the correct vector.
```

TRANS

Law Symbol: Laws

Action: Crates a law that transforms positions.

Derivation: transform_law_data : base_transform_law_data : law_data : ACIS_OBJECT : –

Syntax: **TRANS**

Description: The trans law symbol requires that my_law return positions. It produces positions that have been transformed by the my_transf. rotate is used on vectors, while trans is used to transform positions.

Example:

```
; law "TRANS"
; Create a transform, and then create its inverse.
(define my_trans_rot (transform:rotation
(position 0 0 0) (gvector 1 0 0) 90))
;; my_trans_rot
; => #[transform 1075284160]
(define my_trans_move (transform:translation
(gvector 1 0 0)))
;; my_trans_move
; => #[transform 1075284848]
(define my_t_comp (transform:compose
my_trans_rot my_trans_move))
;; my_t_comp
(define my_law (law "trans(vec(x,y,z),trans1)"
my_t_comp))
;; my_law
; => #[law "TRANS(VEC(X,Y,Z),TRANS1)"]

; This transforms the given law "VEC(X,Y,Z)" by
; the supplied transform, my_t_comp
(law:eval my_law (list 0 0 1))
;; (1 –1 6.12323399573677e–17)
; In this example, the input vector is (0, 0, 1).
; It gets rotated by 90 degrees, causing y to be
; –1 and then gets moved along x axis by 1.
; z is approximately zero.
```

WIRE

Law Symbol: Laws

Action: Creates a law that returns the positions of the wire's component edges.

Derivation: wire_law_data : base_wire_law_data : path_law_data : law_data : ACIS_OBJECT : –

Syntax: **WIRE**

Description: A wire is parameterized from 0 to the length of the wire. This symbol returns the position of the wire's component edges. The parameterization has been linearly scaled to match the total length of the edge.

ACIS parameterization is not the arc length. The wire law returns the position as a function of arc length, in as much linear scaling as the subedges can accomplish. In the case of lines and arcs, the parameterization is exactly the arc length. Curves which are not parameterized with constant speed may have some internal variance. All curves other than arcs and lines have non–constant speed.

Example:

```
; law ”WIRE”
; Create an edge.
(define my_edge (edge:circular
    (position 0 0 0) 20))
;; my_edge
; => #[entity 2 1]
; Create a wire body.
(define my_wire (wire–body my_edge))
;; my_wire
; => #[entity 3 1]
; Input this wire into a law.
(define my_law (law ”wire(wire1)” my_wire))
;; my_law
; => #[law ”WIRE(WIRE1)”]
; Evaluate this law mathematic function at a
; parameter value.
(law:eval–position my_law 2)
;; #[position 19.9000833055605 1.99666833293656 0]
```