Chapter 7.
Save Identifiers R thru Z

“rbase”

Purpose: Defines a rendering attribute that holds information about surface visual attributes, sidedness, and texture space.

Derivation: ATTRIB_RH : ATTRIB : ENTITY : –

Data Elements:

prim	No data	This class does not save any data

Description: This class defines a rendering attribute that holds information about surface visual attributes (RH_MATERIAL*), surface-sidedness for rendering (int), and surface texture space for wrapping (RH_TEXTURE_SPACE*) textures onto the surface.

“rbblnsur”

Purpose: This class implements the constant radius rolling ball blend surface.

Derivation: rb_blend_spl_sur : blend_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:

prim	subtype_start	Left curly braces, “{” or Tag 15
prim	write sv id	save identifier for this particular subtype
sv id	blend_spl_sur	Blend data
prim	subtype_end	Right curly braces, “}” or Tag 16

Description: This is a straightforward derivation of blend_spl_sur. The ball rolls on two support entities, which may be either curves, surfaces or points. The point-point case is not included because this is always a sphere. The surface-surface case is equivalent to the pipe surface.

“real_attrib”

Purpose: Defines a generic attribute that contains a real value.

Derivation: ATTRIB_GEN_REAL : ATTRIB_GEN_NAME : ATTRIB_GENERIC : : ENTITY : – ATTRIB

Data Elements: prim real Value

Description: Refer to the Purpose.

“rem_edge”

Purpose: For internal use only.

Derivation: REM_EDGE : EDGE : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: Refer to Purpose.

“rem_protected_list_att”

Purpose: For internal use only.

Derivation: ATTRIB_REM_PROTECTED_LIST : ATTRIB_SYS : ATTRIB : ENTITY :

Data Elements: prim No data This class does not save any data

Description: Results in an error message “Bad op on att” when in debug mode.

“rem_vertex”

Purpose: For internal use only.

Derivation: REM_VERTEX : VERTEX : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: Refer to Purpose.

“render”

Purpose: Defines an attribute that contains rendering information.

Derivation: ATTRIB_RENDER : ATTRIB_RH : ATTRIB : ENTITY : –

Data Elements:

```
prim    $rec_num                    entity material
prim    $rec_num                    entity tspace
prim    integer                     number of sides
ctrl    if_cond                     if ent_tran is set
ctrl        if_cond                 if tran_mod is set
prim            integer             write the number 2
ctrl        else_cond               if not tran_mod
prim            integer             write the number 1
prim            transf              write the transform for ent_tran
ctrl    else_cond                   if not ent_tran
prim        integer                 write integer 0
```

Description: Refer to the Purpose.

“rgb_color”

Purpose: Defines a ATTRIB_RGB color attribute.

Derivation: ATTRIB_RGB : ATTRIB_ST : ATTRIB : ENTITY : –

Data Elements:

prim	real	red color component
prim	real	green color component
prim	real	blue color component

Description: Defines ATTRIB_RGB to store RGB color information for an ENTITY. This attribute takes precedence over ATTRIB_COL when displaying ENTITYs.

“ref”

Purpose: This references a previously defined subtype in the save file.

Derivation: None

Data Elements:	prim	subtype_start	Left curly braces, “{” or Tag 15
	prim	write sv id	save identifier for this particular subtype
	prim	integer	Subtype index number being referenced.
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: A subtype that has been previously defined can be referenced using this. Thus duplicate information does not have to be stored in the save file. Subtypes receive an index number starting with zero at the beginning of the file. This integer is then later used by a reference to obtain the correct information.

“ref_vt”

Purpose: Attaches REFINEMENT and VERTEX_TEMPLATE instances to other entities.

Derivation: ATTRIB_EYE_REF_VT : ATTRIB_EYE : ATTRIB : –

Data Elements:	prim	$rec_num	Pointer to record in save file for refinement
	prim	$rec_num	Pointer to record in save file for vertex template

Description: The ATTRIB_EYE_REF_VT class is an ACIS attribute used to attach REFINEMENT and VERTEX_TEMPLATE instances to other entities. The class can hold a pointer to one of each class. However, this implementation assumes that exactly one of those pointers is non-NULL. This allows independent replacement, and use of multiple refinements or vertex templates, without undue complication.

“rh_background”

Purpose: Defines a background.

Derivation: RH_BACKGROUND : RH_ENTITY : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: This class defines the color of a pixels at any point in the image which is not covered by an entity surface. A background can comprise a single uniform color or pattern, or can be composed of a previously-generated image or an image scanned from a photograph. Only one background can be active at any one time.

"rh_entity"

Purpose: Provides common methods and data for other rendering classes.

Derivation: RH_ENTITY : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: Rendering entities provide the basis for manipulating the appearance of ACIS geometric entities, image backgrounds and lighting conditions. Child classes include RH_BACKGROUND, RH_FOREGROUND, RH_ENVIRONMENT_MAP, RH_LIGHT, RH_MATERIAL, and RH_TEXTURE_SPACE.

"rh_env_map"

Purpose: Defines an environment map.

Derivation: RH_ENVIRONMENT_MAP : RH_ENTITY : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: Environment maps simulate inter-object reflections, both between bodies in a scene and between a body and the external environment. RH_ENVIRONMENT_MAP objects are used with one of the component shaders of the other rendering entities specified by an RH_MATERIAL.

"rh_foreground"

Purpose: Defines a foreground.

Derivation: RH_FOREGROUND : RH_ENTITY : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: A foreground is the counterpart to a background. Foreground shaders provide an extra level of image processing during the shading process. It can be thought of as a filter and may be used to support atmospheric effects, such as fog or depth cueing. Only one foreground can be active at any given time.

“rh_light”

Purpose: Defines a light source.

Derivation: RH_LIGHT : RH_ENTITY : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: RH_LIGHTs define light sources within the Renderer. Supported light source types are “ambient,” “distant,” “eye,” “point,” and “spot.”

An enumerated type, Fall_Off_Type, is a parameter to some light types which selects how the intensity of the light varies with the distance from the light source, and has possible values of FALL_OFF_CONSTANT, FALL_OFF_INVERSE, or FALL_OFF_INVERSE_SQUARE.

Shadowing is supported for distant, point, and spot in all rendering modes except flat and simple. If an image is to be rendered with shadows, a shadow map must be computed before rendering, using api_rh_create_light_shadow for each light for which shadows are required. A shadow map is view-independent and can be reused for any number of images provided there is no change in the light source geometry or the entities it illuminates.

“rh_material”

Purpose: Defines a material consisting of color, displacement, reflectance, and transparency.

Derivation: RH_MATERIAL : RH_ENTITY : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: A material defines the appearance of the surface of an ACIS topological entity in terms of four components: color, reflectance, transparency, and displacement.

A color defines the color for any point on the surface of an entity to which applies and can be a simple single color or a complex pattern, such as a procedurally-defined marble effect.

The reflectance governs how the surface behaves visually in the presence of light. The reflectance defines the surface finish of an entity and models effects, such as matte, metal, or mirrored surfaces. Reflectance is not supported in the flat or gouraud rendering modes.

"rh_texture_space"

Purpose: Defines a texture space.

Derivation: RH_TEXTURE_SPACE : RH_ENTITY : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: RH_TEXTURE_SPACE entities assist in the production of a shading effect known as a wrapped texture. A wrapped texture produces the effect of a sheet of paper shrink wrapped onto the surface of a solid object. A texture space uses one of several texture space shaders to map between the coordinate system of the sheet and the coordinate system of the surface of the solid object. Texture space arguments are treated in a similar fashion to those of material components.

"ROTATE"

Purpose: Composes a law mathematic function that transforms vectors.

Derivation: rotate_law : multiple_data_law : law : –

Data Elements:	prim	string	The word "ROTATE" followed by something in parenthesis appears somewhere within this double quoted string.

Description: The rotate law symbol requires that my_law return vectors. It produces vectors that have by transformed by the my_transf. rotate is used on vectors, while trans is used to transform positions. If the transform input to this law does a rotation (e.g., transform:rotation) and a translation (e.g., transform:translation), this law only works on the rotational component.

“round”

Purpose: Defines a circular rolling-ball blend.

Derivation: ATTRIB_ROUND : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:

prim	real	Radius
ctrl	if_cond	if save_version_number is greater than or equal to BLEND_VERSION
prim	real	Setback at start
prim	real	Setback at end
prim	real	Bulge

Description: Refer to the Purpose.

“rotsur”

Purpose: Represents a surface of rotation.

Derivation: rot_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start		Left curly braces, “{” or Tag 15
	prim	write sv id		save identifier for this particular subtype
	ctrl	if_cond		if save_version_number is less than SPLINE_VERSION
	sv id		spl_sur	save data
	ctrl	else		
	prim		newline	
	sv id		curve type	Curve being rotated
	prim		newline	
	prim		position	Root of axis
	prim		vector	Direction of axis
	prim		interval	*u* range
	prim		interval	*v* range
	ctrl	if_cond		if save_version_number is greater than or equal to DISCONTINUITY_VERSION
	prim		newline	
	prim		discontinuity_info	U Parameter values of discontinuities
	prim		newline	
	prim		discontinuity_info	V Parameter values of discontinuities
	prim	subtype_end		Right curly braces, “}” or Tag 16

Description: This class represents a surface of rotation. The surface is defined by an axis of rotation and a curve. The curve must not intersect with the axis, except possibly at its ends, and must not be tangential to a circle centered on the axis and perpendicular to it (i.e., at no point on the curve can the tangent direction be the same as that of a circle that is centered on the axis of revolution, perpendicular to it, and through the point). The parameter ranges defining the surface are the *u*-direction is along the curve, and follows its parameterization, while the *v*-direction is clockwise around the axis, with the given curve as the *v*=0 parameter line.

“ruledtapersur”

Purpose: Class to describe a surface tapered about an edge by a constant angle relative to a draft angle.

Derivation: ruled_tpr_spl_sur : edge_tpr_spl_sur : taper_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start	Left curly braces, “{” or Tag 15
	prim	write sv id	save identifier for this particular subtype
	sv id	“edgetapersur”	Save the information from the edge_tpr_spl_sur.
	prim	real	sine angle
	prim	real	cosine angle
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: Class to describe a ruled–tapered surface, in which a surface is tapered about an edge by a constant angle relative to a draft angle. The surface is a ruled surface between two *u* parameter curves.

“SEC”

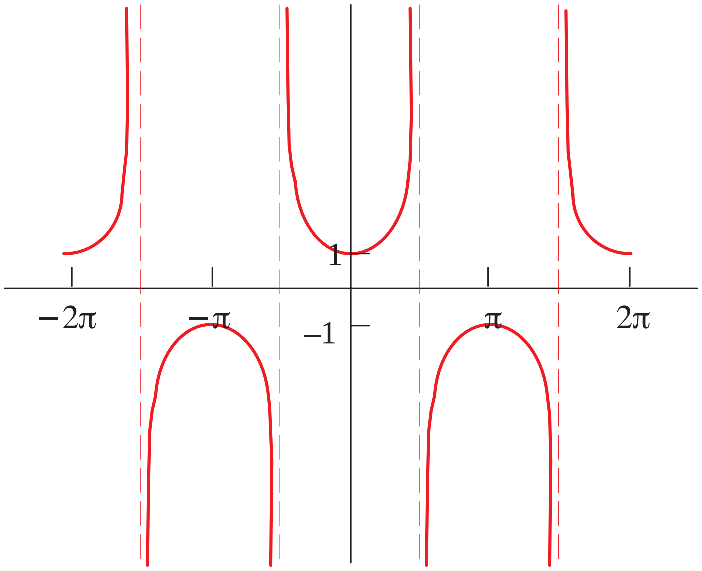
Purpose: Composes a law mathematic function that finds the secant.

Derivation: sec_law : unary_law : law : –

Data Elements:	prim	string	The word “SEC” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$$y = \sec x = \frac{1}{\cos x}$$



"SECH"

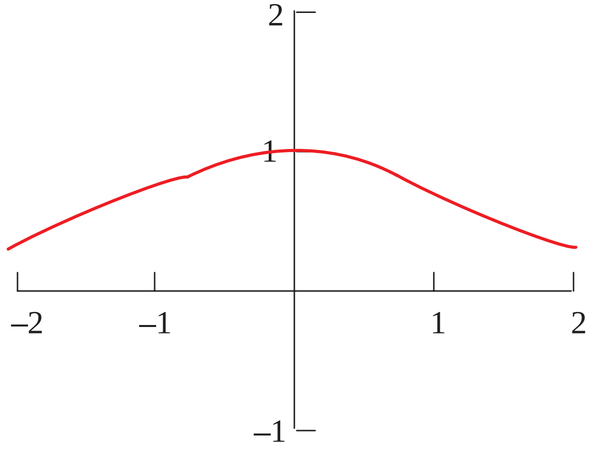
Purpose: Composes a law mathematic function that finds the hyperbolic secant.

Derivation: sech_law : unary_law : law : –

Data Elements:	prim	string	The word "SECH" followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$$y = \operatorname{sech} x = \frac{2}{e^{x} + e^{-x}}$$



“SET”

Purpose: Composes a law mathematic function that returns a 1 if its sublaw is positive and 0 if its sublaw is negative or zero (0).

Derivation: set_law : unary_law : law : –

Data Elements: prim string The word “SET” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The set function is used primarily to define derivatives of special functions. If the sublaw symbol is positive, it returns a 1; if the sublaw symbol is negative, it returns a 0.

For example, the derivative of the absolute value of x is 1 for positive values of x and –1 for negative values of x. Hence, the derivative can be expressed as “set(x)–(1–set(x))”.

The functions abs, max, and min all use the set function when a derivative is taken of them.

"sfcvfreeblndsur"

Purpose: Implements the variable-radius surface-curve/free blend surface.

Derivation: sfcv_free_bl_spl_sur : var_blend_spl_sur : blend_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:	prim	No data	This class does not save any data

Description: This class implements the surface geometry of a variable-radius blend between a curve in a surface and another surface. THe blend is tangent to the surface containing the surface curve, and is not tangent to the other surface.

"sg"

Purpose: Organization base attribute class for the SG Husk.

Derivation: ATTRIB_SG : ATTRIB : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: ATTRIB_SG is the organizational class from which the other SG Husk attribute classes are derived. Its sole purpose is to identify those child classes as belonging to the SG Husk, and so adds no new data or methods to those of ATTRIB.

"shadowtapersur"

Purpose: Class to describe a surface that is tapered about a silhouette by a constant angle determined by a draft direction.

Derivation: shadow_tpr_spl_sur : taper_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start	Left curly braces, “{” or Tag 15
	prim	write sv id	save identifier for this particular subtype
	sv id	“tapersur”	Save the information from the taper_spl_sur.
	prim	vector	draft of the taper
	prim	real	sine angle
	prim	real	cosine angle
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: Class to describe a shadow–tapered surface, in which a surface is tapered about a silhouette by a constant angle determined by a draft direction.

“shell”

Purpose: Bounds a LUMP peripherally, or as an internal void.

Derivation: SHELL : ENTITY : –

Data Elements:	prim	$rec_num	Pointer to record in save file for next shell in lump
	prim	$rec_num	Pointer to record in save file for first subshell in shell
	prim	$rec_num	Pointer to record in save file for first face in shell
	ctrl	if_cond	if save_version_number is greater than or equal to WIREBOOL_VERSION
	prim	$rec_num	Pointer to record in save file for first wire in shell
	ctrl	if_cond	if save_version_number is less than LUMP_VERSION
	prim	$rec_num	Pointer to record in save file for body owning the LUMP containing shell
	ctrl	else	
	prim	$rec_num	Pointer to record in save file for lump containing shell

Description: The SHELL is one portion of a LUMP’s boundary, and has no internal connection with any other SHELL. If a LUMP has no voids, exactly one *peripheral* SHELL gives it its overall extent; any other SHELLs bound *voids* wholly within the LUMP. In the data structure, no distinction is made between peripheral and void SHELLs. In this context a SHELL is closed and bounded.

It is technically possible for a SHELL to be open and bounded or unbounded. If bounded, the containing LUMP (and BODY) is considered *incomplete*, more accurately, it is incompletely bounded. It interacts with other BODYs only so far as the defined portions of their SHELLs interact, and there are configurations of that interaction that are disallowed. If the SHELL is unbounded, it can be semi-infinite (e.g., a plane bounded by a single infinite straight line) or infinite (two half-infinite planes joined at their boundaries). If the SHELL is semi-infinite, the BODY is incomplete, while an infinite SHELL is completely defined, though of infinite extent.

The concepts of *peripheral* and *void* SHELLs, and of *connected* and *disjoint* BODYs have no meaning when applied to incomplete LUMPs or BODYs.

A SHELL is constructed from a collection of WIREs or FACEs. If this is a large collection, it may be subdivided into a hierarchy of SUBSHELLs, each containing a proper subcollection. A SHELL subdivided into SUBSHELLs may also contain WIREs and FACEs directly, if these entities do not fit naturally into any SUBSHELL.

“SIN”

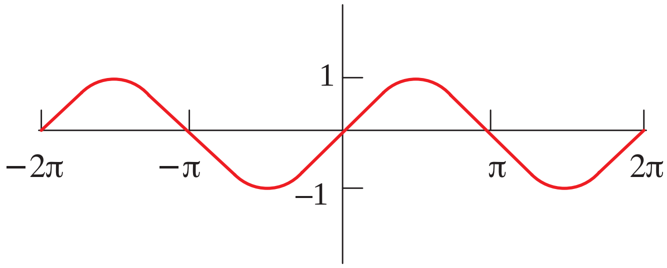
Purpose: Composes a law mathematic function that finds the sine.

Derivation: sin_law : unary_law : law : –

Data Elements:	prim	string	The word “SIN” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$y = \sin x$



“SINH”

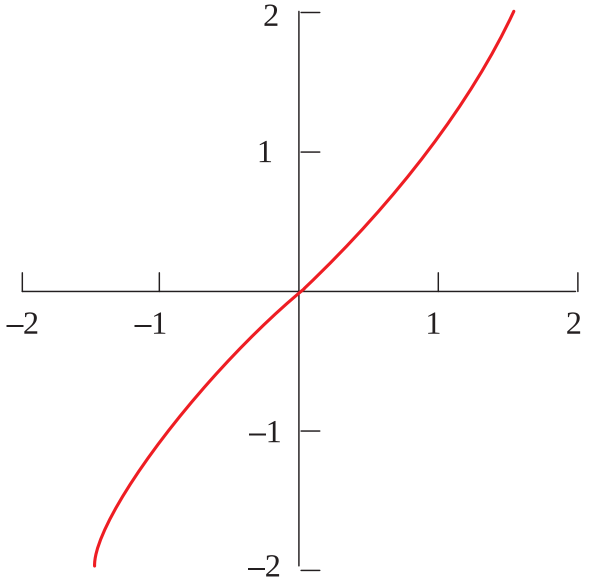
Purpose: Composes a law mathematic function that finds the hyperbolic sine.

Derivation: sinh_law : unary_law : law : –

Data Elements:	prim	string	The word “SINH” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$$y = \sinh x = \frac{e^{x} - e^{-x}}{2}$$



"SIZE"

Purpose: Returns the square root of the sum of the squares of a given vector (e.g., VEC) elements.

Derivation: size_law : unary_law : law : –

Data Elements: prim	string	The word "SIZE" followed by something in parenthesis appears somewhere within this double quoted string.

Description: Refer to Purpose.

"skinsur"

Purpose: Defines a skin surface between a list of curves.

Derivation: skin_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start	Left curly braces, “{” or Tag 15
	prim	write sv id	save identifier for this particular subtype
	prim	integer	Number of curves
	ctrl	repeat	Repeat for the number of curves
	prim	real	Tangent length at start of curve or –1
	prim	real	Tangent length at end of curve or –1
	prim	real	Matching tangent length at start of curve or –1
	prim	real	Matching tangent length at end of curve or –1
	prim	real	*v* knot
	sv id	curve type	Curve to be skinned/lofted
	prim	vector	*v* derivative or zero vector
	sv id	surface type	Surface
	prim	real	Tangent factor or 0
	sv id	spl_sur	Generic spl_sur data
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: This class defines a skin surface between a list of curves.

Surface Parameterization

The surface parameterization is the *u*-direction corresponds to the parameterization of the curves to be skinned and the *v*-direction corresponds to the cubic Bezier between the skin-curves.

The input to this surface class are the curves to be skinned (all the curves are reparameterized to lie in [0.0 – 1.0] range), optional tangents (the magnitude of the curves’ tangents have to match on the ends) in *u*-direction, and the optional surfaces on which the curves lie. If surfaces containing the curves are provided, these determine the tangent directions in *v*.

Evaluation Process

The evaluation process is a three-step process.

1. If any matching tangent magnitudes are given, the section curves (curves to be skinned) are reparameterized as follows: parameter *t* is the parameter on the original curve. Parameter *u* on the skin surface is determined such that the *u* partial at each end of the skin surface is equal to the matching tangent magnitude.

2. The tangent directions for the *v* are determined by fitting a circle through the points corresponding to the same *u* value on the adjacent section curves to the left and right. The Scheme followed is similar to the way Bessel tangents are computed. If there are only two section curves, the circle radius is chosen to be infinity. If the surfaces are given for any section, the tangent direction in *v* when on that curve is obtained by the cross product of surface normal and the section curve tangent at that point. The direction also has an optional scalar value that can be applied. The surface is called a loft surface when such a surface is provided.

3. Now the skin/loft surface is defined using Hermite interpolants between sections that join each other C1 continuously. To evaluate the surface S(*u*,*v*) at a particular *v*-parameter, the first step is to find the segment to which this parameter corresponds. Then a local parameter v_i is computed, which ranges from 0 to 1. The section curves c_i and c_{i+1}, and the tangents s_i and t_{i+1} are also obtained.

The parametric derivatives of this surface are obtained by differentiating the above equation algebraically.

“skinsur2”

Purpose: Defines a skin surface between a list of curves.

Derivation: skin_spl_sur2 : spl_sur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start			Left curly braces, “{” or Tag 15
	prim	write sv id			save identifier for this particular subtype
	ctrl	if_cond			if the save version number is less than the SPLINE_VERSION
	sv id		spl_sur		save the spline surface data
	ctrl	else			if the save version number is greater than the SPLINE_VERSION
	prim		integer		Number of curves
	prim		newline		Newline for readability
	ctrl		repeat		Repeat for the number of curves
	sv id			curve type	Curve to be skinned/lofted
	prim			newline	Newline for readability
	sv id		spl_sur		Generic spl_sur data
	prim	subtype_end			Right curly braces, “}” or Tag 16

Description: This class defines a skin surface between a list of curves.

“sphere”

Purpose: Identifier used by more than one class.

Derivation: None

Data Elements:	ctrl	if_cond		if not a subtype reference; save identifier appended to beginning of record, while its data is appended to the end of the record.
	sv id		SPHERE (1) class	derived from SPHERE class
	ctrl	else		it is a subtype reference; save identifier is followed immediately by its data, both enclosed by subtype_start and subtype_end.
	sv id		sphere (2) class	derived from sphere class

Description: Used to determine which class specified the sphere. A subtype reference is inline with a definition and is surrounded by curly braces { }, or Tag 15 and 16.

SPHERE (1) class

Purpose: Defines a sphere as an object in the model.

Derivation: SPHERE : SURFACE : ENTITY : –

Data Elements:	sv id	sphere (2) class	sphere data given in another section of this manual.

Description: A SPHERE is defined by sphere data, which includes a radius and position. The sign of the radius determines the direction of the surface. A positive radius defines a convex sphere, a negative radius a concave sphere.

sphere (2) class

Purpose: Defines a spherical surface.

Derivation: sphere : surface : –

Data Elements:

ctrl	if_cond	if used as a subtype reference
prim	subtype_start	Left curly braces, “{” or Tag 15
ctrl	if_cond	if save_version_number is less than the SURFACE_VERSION
prim	integer	sphere_type; integer for type of sphere
ctrl	else	if save_version_number is greater than the SURFACE_VERSION
prim	string	save identifier; “sphere”.
prim	position	center of sphere
prim	real	radius of sphere
ctrl	if_cond	if save_version_number is greater than or equal to SURFACE_VERSION
prim	vector	*uv* origin direction
prim	vector	pole direction
prim	logical	Either “forward_v” or “reverse_v”
sv id	surface (2) class	Generic surface data. Refer to another section of this manual.
ctrl	if_cond	if used as a subtype reference
prim	subtype_end	Right curly braces, “}” or Tag 16

Description:

A sphere is defined by a center point and radius. Normally, the radius is positive (i.e., material inside the surface), but it can be negative, indicating a “hollow” sphere (i.e., material outside the surface).

The *u*-parameter is the latitude metric, running from –pi/2 at the south pole through 0 at the equator to pi/2 at the north pole. The *v*-parameter is the longitude metric, running from –pi to pi, with 0 on the meridian containing ori_dir, and increasing in a clockwise direction around pole_dir, unless reverse_v is TRUE.

Let *P* be pole_dir and *Q* ori_dir, and let *R* be *P* x *Q*, negated if reverse_v is TRUE. Let *r* be the absolute value of the sphere radius. Then:

```
pos = center + r* sin(u)* P + r* cos(u)*
    (cos(v)* Q + sin(v) R)
```

This parameterization is left-handed for a convex sphere and right-handed for a hollow one, if reverse_v is FALSE, and reversed if it is TRUE.

When the sphere is transformed, the sense of reverse_v is inverted if the transform includes a reflection. No special action is required for a negation.

In summary, spheres are:

- – Not true parametric surfaces.
- – Periodic in *v* (–pi to pi with period 2pi) but not in *u*.
- – Closed in *v* but not in *u*.
- – Singular in *u* at the poles; non-singular everywhere else.

“spline”

Purpose: Identifier used by more than one class.

Derivation: None

Data Elements:	ctrl	if_cond	if not a subtype reference; save identifier appended to beginning of record, while its data is appended to the end of the record.
	sv id	SPLINE (1) class	derived from SPLINE class
	ctrl	else	it is a subtype reference; save identifier is followed immediately by its data, both enclosed by subtype_start and subtype_end.
	sv id	spline (2) class	derived from spline class

Description: Used to determine which class specified the spline. A subtype reference is inline with a definition and is surrounded by curly braces { }, or Tag 15 and 16.

SPLINE (1) class

Purpose: Defines a parametric surface as an object in the model.

Derivation: SPLINE : SURFACE : ENTITY : –

Data Elements:	sv id	spline (2) class	Spline surface definition

Description: A SPLINE records a parametric surface as a spline. In turn, a SPLINE holds a pointer to a spl_sur and a logical denoting reversal of the sense of the stored surface.

spline (2) class

Purpose: Records a B-spline surface.

Derivation: spline : surface : –

Data Elements:

ctrl	if_cond			if used as a subtype reference
prim		subtype_start		Left curly braces, “{” or Tag 15
ctrl		if_cond		if save_version_number is less than the SURFACE_VERSION
prim			integer	spline type
ctrl		else		
prim			save ID	spline’s save ID
ctrl	if_cond			if save_version_number is less than SPLINE_VERSION
ctrl		case_cond		specific surface definition subtype
sv id			“exactsur”	if spl_sur is exact_spl_sur or “exact_sur”
sv id			“offsur”	if spl_sur is off_spl_spl_sur or “offsur”
sv id			“pipesur”	if spl_sur is pipe_spl_sur or “pipesur”
sv id			“rbblnsur”	if spl_sur is rb_blend_spl_sur or “rbblsur”
sv id			“rotsur”	if spl_sur is rot_spl_sur or “rotsur”
sv id			“skinsur”	if spl_sur is skin_spl_sur or “skinsur”
sv id			“subsur”	if spl_sur is sub_spl_sur or “subsur”
sv id			“sumsur”	if spl_sur is sum_spl_sur or “sum_sur”
sv id			“sweepsur”	if spl_sur is sweep_spl_sur or “sweep_sur”
sv id			“tubesur”	if spl_sur is tube_spl_sur or “tube_sur”
sv id			“vertexblendsur”	if spl_sur is VBL_SURF or “vertexblendsur”
sv id			“varblendsplsur”	if spl_sur is exact_spl_sur or “exact_sur”
ctrl	else			
prim		logical		“forward” or “reversed”

ctrl	case_cond	specific surface definition subtype
sv id	“exactsur”	if spl_sur is exact_spl_sur or “exact_sur”
sv id	“offsur”	if spl_sur is off_spl_spl_sur or “offsur”
sv id	“pipesur”	if spl_sur is pipe_spl_sur or “pipesur”
sv id	“rbblnsur”	if spl_sur is rb_blend_spl_sur or “rbblsur”
sv id	“rotsur”	if spl_sur is rot_spl_sur or “rotsur”
sv id	“skinsur”	if spl_sur is skin_spl_sur or “skinsur”
sv id	“subsur”	if spl_sur is sub_spl_sur or “subsur”
sv id	“sumsur”	if spl_sur is sum_spl_sur or “sum_sur”
sv id	“sweepsur”	if spl_sur is sweep_spl_sur or “sweep_sur”
sv id	“tubesur”	if spl_sur is tube_spl_sur or “tube_sur”
sv id	“vertexblendsur”	if spl_sur is VBL_SURF or “vertexblendsur”
sv id	“varblendsplsur”	if spl_sur is exact_spl_sur or “exact_sur”
sv id	surface (2) class	Generic surface data
ctrl	if_cond	if used as a subtype reference
prim	subtype_end	Right curly braces, “}” or Tag 16

Description: The spline class represents a parametric surface that maps a rectangle within a 2D real vector space (parameter space) into a 3D real vector space (object space). This mapping must be continuous, and one-to-one except possibly at the boundary of the rectangle in parameter space. It is differentiable twice, and the normal direction is continuous, though the derivatives need not be. The positive direction of the normal is in the sense of the cross product of the partial derivatives with respect to *u* and *v* in that order. The portion of the neighborhood of any point on the surface that the normal points to is outside the surface, and the other part is inside.

Opposite sides of the rectangle can map into identical lines in object space, in which case the surface is closed in the parameter direction normal to those boundaries. If the parameterization and derivatives also match at these boundaries, the surface is periodic in this parameter direction. The line in object space corresponding to the coincident boundaries is known as the seam of a periodic surface.

If a surface is periodic in one parameter direction, it is defined for all values of that parameter. A parameter value outside the domain rectangle is brought within the rectangle by adding a multiple of the rectangle’s width in that parameter direction, and the surface evaluated at that value. If the surface is periodic in both parameters, it is defined for all parameter pairs (*u*,*v*), with reduction to standard range happening with both parameters.

One side of the rectangle can map into a single point in object space. This point is a parametric singularity of the surface. If the surface normal is not continuous at this point, it is a surface singularity.

The spline contains a “reversed” bit together with a pointer to another structure, a spl_sur or something derived from it, that contains the bulk of the information about the surface.

“split”

Purpose: Attached to each edge of each body which has a graph vertex properly within it.

Derivation: ATTRIB_SPLIT : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:		
prim	No data	This class does not save any data

Description: For internal use only. This attribute is attached to each edge of each affected body which has a graph vertex properly within it. It simply contains a list of all the graph vertices, in order along the edge, so that when the edge is split at one of them, the others may be associated with the correct piece for later splitting.

spl_sur

Purpose: Defines an abstract base class from which spline surface definitions are derived.

Derivation: spl_sur : subtrans_object : subtype_object : –

Data Elements:

sv id	bs3_surface_def	B-spline approximation of surface
prim	real	Fit tolerance
ctrl	if_cond	if save_version_number is greater than or equal to DISCONTINUITY_VERSION
prim	newline	
prim	discontinuity_info	U Parameter values of discontinuities
prim	newline	
prim	discontinuity_info	V Parameter values of discontinuities

Description: In ACIS a sculptured surface is represented by the class spline, which contains a pointer to an internal description called spl_sur. The spl_sur further contains a bs3_surface that is a pointer to a rational or nonrational, nonuniform B-spline surface in the underlying surface package.

“spring”

Purpose: Marks edges lying on spring curves so they may be specially handled later.

Derivation: ATTRIB_SPRING : ATTRIB_BLINFO : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:

prim	No data	This class does not save any data

Description: This class marks edges lying on spring curves so they may be specially handled later. The attribute refers to the face of the blended edge from which the blend springs.

"spring_load"

Purpose: For internal use only.

Derivation: ATTRIB_DS_SPRING : ATTRIB_DSLOAD : ATTRIB : ENTITY : –

Data Elements:

prim	long		asp_slide_state
prim	long		asp_domain_dim
ctrl	repeat		Repeat for all asp_domain_dim
prim		real	asp_domain_pt[ii]
prim	long		asp_image_dim
ctrl	repeat		Repeat for all asp_image_dim
prim		real	asp_free_pt[ii]

Description: Refer to Purpose.

"spring_set_load"

Purpose: Set of spring loads.

Derivation: ATTRIB_SPRING_SET : ATTRIB_DSLOAD : ATTRIB : ENTITY : –

Data Elements:

prim	long		ass_pt_count
prim	long		ass_domain_dim
prim	long		ass_image_dim
ctrl	repeat		Repeat for all ass_pt_count * ass_domain_dim
prim		real	ass_domain_pt[ii]
prim	repeat		Repeat for all ass_pt_count * ass_image_dim
prim		real	ass_free_pt[ii]

Description: For internal use only.

"SQRT"

Purpose: Composes a law mathematic function that takes the square root of a given law.

Derivation: sqrt_law : unary_law : law : –

Data Elements: prim string The word “SQRT” followed by something in parenthesis appears somewhere within this double quoted string.

Description: Refer to Purpose statement.

“srfsrfblndsur”

Purpose: Implements the variable-radius face-face blend surface.

Derivation: srf_srf_v_bl_spl_sur : var_blend_spl_sur : blend_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements: prim No data This class does not save any data

Description: This class implements the surface geometry of a variable radius blend between two surfaces. The blend will be tangent to both surfaces.

“st”

Purpose: Organization attribute from which various color, display, id, and other attributes are derived.

Derivation: ATTRIB_ST : ATTRIB : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: This class is an attribute declaration for a private container attribute. This class derives from the ACIS base class, ATTRIB. Each application developer receives a customized attribute declaration. The application developer then makes all attributes specific to the application-derived classes of this attribute, ensuring that different developers can assign identifiers independently without interference.

“STEP”

Purpose: Composes a law mathematic function that defines functions with disjoint intervals.

Derivation: step_law : multiple_law : law : –

Data Elements:	prim	string	The word “STEP” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The step law symbol is an array alternating laws and numbers. The numbers divide the real line into disjoint intervals: from minus infinity to num1, num1 to num2, and numx to positive infinity. A later evaluation uses my_law1 for the first interval, my_law2 for the second, etc.

When evaluating a step symbol at its boundaries, the second law has precedence. If we have the law defined by ”step(1, 0, 2*x, 1, –1)” and we evaluate it at x=1, the answer is –1 rather than 2.

“sti_elat_attr”

Purpose: Creates a temporary attribute used in sweeping.

Derivation: ATTRIB_STI_ELAT_ATTR : ATTRIB_SG : ATTRIB : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: For internal use only.

“sti_nor_attr”

Purpose: Creates a temporary attribute used in sweeping.

Derivation: ATTRIB_STI_NOR_ATTR : ATTRIB_SG : ATTRIB : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: For internal use only.

“sti_prof_attr”

Purpose: Creates a temporary attribute used in sweeping.

Derivation: ATTRIB_STI_PROF_ATTR : ATTRIB_SG : ATTRIB : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: For internal use only.

“sti_psplit_attr”

Purpose: Creates a temporary attribute used in sweeping.

Derivation: ATTRIB_STI_PSPLIT_ATTR : ATTRIB_SG : ATTRIB : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: For internal use only.

“sti_rel_attr”

Purpose: Creates a temporary attribute used in sweeping.

Derivation: ATTRIB_STI_REL_ATTR : ATTRIB_SG : ATTRIB : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: For internal use only.

“sti_sect_attr”

Purpose: Creates a temporary attribute used in sweeping.

Derivation: ATTRIB_STI_PSPLIT_ATTR : ATTRIB_SG : ATTRIB : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: For internal use only.

“sti_vlat_attr”

Purpose: Creates a temporary attribute used in sweeping.

Derivation: ATTRIB_STI_VLAT_ATTR : ATTRIB_SG : ATTRIB : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: For internal use only.

“straight”

Purpose: Identifier used by more than one class.

Derivation: None

Data Elements:	ctrl	if_cond	if not a subtype reference; save identifier appended to beginning of record, while its data is appended to the end of the record.
	sv id	STRAIGHT (1) class	derived from STRAIGHT class
	ctrl	else	it is a subtype reference; save identifier is followed immediately by its data, both enclosed by subtype_start and subtype_end.
	sv id	straight (2) class	derived from straight class

Description: Used to determine which class specified the straight. A subtype reference is inline with a definition and is surrounded by curly braces { }, or Tag 15 and 16.

STRAIGHT (1) class

Purpose: Defines an infinite line as an object in the model.

Derivation: STRAIGHT : CURVE : ENTITY : –

Data Elements:	sv id	straight (2) class	Line definition

Description: A STRAIGHT is defined by a point (position) on an infinite line and its direction (unit_vector).

straight (2) class

Purpose: Defines an infinite straight line represented by a point and a unit vector specifying the direction.

Derivation: straight : curve : –

Data Elements:

ctrl	if_cond			if used as a subtype reference
prim		subtype_start		Left curly braces, “{” or Tag 15
ctrl		if_cond		if save_version_number is less than the CURVE_VERSION
prim			integer	straight type
ctrl		else		
prim			string	save identifier; “straight”
prim	position			Root point
prim	vector			Direction
sv id	curve (2) class			Generic curve data
ctrl	if_cond			if used as a subtype reference
prim		subtype_end		Right curly braces, “}” or Tag 16

Description: This class defines an infinite straight line represented by a point and a unit vector specifying the direction. A straight also has a scale factor for the parameterization, so the parameter values can be made invariant under transformation. A straight line is an open curve that is not periodic. It is parameterized as:

```
point = root_point + t* param_scale* direction
```

where t is the parameter.

“string_attrib”

Purpose: Defines a generic attribute that contains a string value.

Derivation: ATTRIB_GEN_STRING : ATTRIB_GEN_NAME : ATTRIB_GENERIC : ATTRIB : ENTITY : –

Data Elements:

ctrl	if_cond		if value is not equal to NULL
prim		string	Value
ctrl	else		
prim		NULL	indicator for empty string

Description: Refer to the Purpose.

“stripc”

Purpose: Identifier used by more than one class.

Derivation: None

Data Elements:	ctrl	if_cond		if not a subtype reference; save identifier appended to beginning of record, while its data is appended to the end of the record.
	sv id		STRIPC (1) class	derived from STRIPC class
	ctrl	else		it is a subtype reference; save identifier is followed immediately by its data, both enclosed by subtype_start and subtype_end.
	sv id		stripc (2) class	derived from stripc class

Description: Used to determine which class specified the stripc. A subtype reference is inline with a definition and is surrounded by curly braces { }, or Tag 15 and 16.

STRIPC (1) class

Purpose: Records a parametric surface as a STRIPC.

Derivation: STRIPC : SURFACE : ENTITY : –

Data Elements:	sv id	stripc (2) class	Strip curve definition

Description: A class derived from SURFACE, records a parametric surface as a (lowercase) stripc.

7

stripc (2) class

Purpose: The strip curve (stripc) is a surface defined in a neighborhood of and passing through a given object-space curve.

Derivation: stripc : surface : –

Data Elements:	ctrl	if_cond	if used as a subtype reference
	prim	subtype_start	Left curly braces, “{” or Tag 15
	ctrl	if_cond	if save_version_number is less than the SURFACE_VERSION
	prim	integer	stripc_type; integer for type of stripc type
	ctrl	else	if save_version_number is greater than the SURFACE_VERSION
	prim	string	save identifier; “stripc”.
	sv id	curve type	Object space curve
	sv id	surface type	Surface on which curve lies
	prim	boolean	Parameter space curve reversed
	sv id	bs2_curve_def	Parameter space curve
	prim	real	Fit tolerance for parameter space curve
	prim	newline	
	prim	logical	*v*-parameter: “forward_v” or “reversed_v” with respect to right hand rule
	sv id	surface (2) class	Generic surface data
	ctrl	if_cond	if used as a subtype reference
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: The strip curve (stripc) is a surface defined in a neighborhood of and passing through a given object-space curve, which is everywhere perpendicular to a given surface in which the curve lies. The surface is a strip two resabs wide, centered on the curve and normal to the surface the strip curve lies in. It is only guaranteed to be well behaved in a neighborhood of the curve. It is used for giving a sense to a curve lying in a surface, allowing portions of the surface on either side of the curve to be distinguished. It may not be used as the surface of a FACE.

The parameterization of the strip curve is determined by the object-space curve and an additional item reverse_v.

Given a surface parameter value (*u*,*v*), the underlying curve is evaluated at parameter u, to give position *P* and first derivative *U*. The underlying surface normal is then obtained, and negated if reverse_v is TRUE, giving *N*. Then the evaluated position is:

```
pos = P + v * |U| * N
```

This is significant only for infinitesimal values of *v*, but demonstrates how derivatives can be defined. In this implementation, second derivatives and curvatures are not accurate for nonzero *v*-parameter; i.e., off the defining curve and surface.

When a strip curve is transformed, its underlying curve and surface are transformed, and then, if the transformation includes reflection, the surface is negated and reverse_v inverted. Negate the strip curve, negate the supporting surface and invert reverse_v.

“subsetintcur”

Purpose: Represents a subset of a longer curve.

Derivation: subset_int_cur : int_cur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start	Left curly braces, “{” or Tag 15
	prim	write sv id	save identifier for this particular subtype
	sv id	int_cur	Generic int_cur data
	enum	CURVE_EXTENSION_TYPE	Extension type
	sv id	curve type	
			Original curve
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: Refer to the Purpose.

“subshell”

Purpose: Represents a subdivision of a SHELL or SUBSHELL.

Derivation: SUBSHELL : ENTITY : –

Data Elements:	prim	$rec_num	Pointer to record in save file for parent subshell
	prim	$rec_num	Pointer to record in save file for next subshell belonging to parent
	prim	$rec_num	Pointer to record in save file for first child subshell
	prim	$rec_num	Pointer to record in save file for first face in subshell
	prim	$rec_num	Pointer to record in save file for first wire in subshell

Description: A subshell represents a subdivision of a SHELL or SUBSHELL. It allows groups of WIREs and FACEs to be excluded by a single box test and improves the efficiency of many-to-many comparisons. The subdivision is determined by the system, and may change at any time. The SUBSHELL has no significance to the end user (the application programmer may find the implied spatial subdivision useful).

“subsur”

Purpose: Represents the geometry of a spline surface, which is a subset region of another spl_sur.

Derivation: sub_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:

prim	subtype_start	Left curly braces, “{” or Tag 15
prim	write sv id	save identifier for this particular subtype
prim	interval	*u* range
prim	interval	*v* range
sv id	surface type	Spline, original surface
prim	subtype_end	Right curly braces, “}” or Tag 16

Description: This class represents the geometry of a spline surface, which is a subset region of another spl_sur. The subset *uv* range may be smaller or larger than the range of the progenitor surface, or it may overlap it.

“sumsur”

Purpose: Represents a linear sum of two curves.

Derivation: sum_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start	Left curly braces, “{” or Tag 15
	prim	write sv id	save identifier for this particular subtype
	ctrl	if_cond	if save_version_number is less than SPLINE_VERSION
	sv id	spl_sur	save data
	ctrl	else	
	prim	newline	
	sv id	curve type	*u* curve
	prim	newline	
	sv id	curve type	*v* curve
	prim	newline	
	prim	position	Datum point
	prim	newline	
	prim	interval	*u* range
	prim	interval	*v* range
	ctrl	if_cond	if save_version_number is greater than or equal to DISCONTINUITY_VERSION
	prim	newline	
	prim	discontinuity_info	U Parameter values of discontinuities
	prim	newline	
	prim	discontinuity_info	V Parameter values of discontinuities
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: This class represents a surface that is a linear sum of two curves. This is derived from the class spl_sur, which is used by the spline surface class to contain the surface descriptions. The surface is defined primarily by two curves that are assumed not parallel, and the parameter ranges over which the surface is defined.

Parametric Representation
If the curves are represented as:

$$x = c1(t) \quad \text{and} \quad x=c2(t)$$

respectively, the surface is:

$$x=s(u,v) = c1(u) + c2(v) - p$$

where *p* is a constant position, normally initialized to be the value of *c2* at the start of the parameter range.

subtype_object

Purpose: Defines the master object from which all subtype objects must be derived.

Derivation: subtype_object : –

Data Elements:

prim	No data	This class does not save any data

Description: This class defines the master object from which all subtype objects must be derived. This object contains a use count (in case the object is shareable) and defines two virtual functions and a destructor.

“supercell”

Purpose: Identifies a grouping of cells or inferior supercells.

Derivation: SUPERCELL : ENTITY : –

Data Elements:

prim	$rec_num	Pointer to record in save file for parent SUPERCELL
prim	$rec_num	Pointer to record in save file for next SUPERCELL (sibling)with same parent
prim	$rec_num	Pointer to record in save file for first child SUPERCELL
prim	$rec_num	Pointer to record in save file for first cell contained in SUPERCELL

Description: This represents a grouping of cells or inferior supercells. It allows the system to improve the efficiency of many-to-many comparisons, by allowing quantities of cells to be excluded by a single box test. The subdivision is determined by the system, and may change at any time, so the supercell has no significance to the user (though the application program may find the spatial subdivision implied useful).

“SURF”

Purpose: Composes a law mathematic function that returns the positions of the defining surface.

Derivation: surface_law : unary_data_law : law : –

Data Elements:	prim	string		The word “SURF” followed by something in parenthesis appears somewhere within this double quoted string.

Description: surf returns the positions of the defining surface at the parameter value. In other words, 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.

“SURF#”

Purpose: Composes a law function with a tag for a surface used as an input argument.

Derivation: surface_law_data : law_data : –

Data Elements:	prim	string			The word “SURF” followed by an integer appears somewhere within this double quoted string.
	prim	integer			An integer greater than 0. It indicates how many law data support items there are.
	ctrl	repeat			Repeat the next steps for each law data support item.
	prim		string		This is a double quoted string with one of the words: “TRANS”, “EDGE”, “SURF”, or “WIRE”.
	ctrl		if_cond		If the string is the double quoted ”SURF”
	sv id			surface type	Save the underlying curve for this edge.
	prim			interval	*u* space interval for the surface
	prim			interval	*v* space interval for the surface

Description: When a surface is used as input into a law function , it is always followed by an integer *n* that specifies its index into the input argument list. The index numbering starts at 1. For any given index number *n*, the argument list has to contain at least *n* arguments.

“surface”

Purpose: Identifier used by more than one class.

Derivation: None

Data Elements:

ctrl	if_cond		if not a subtype reference; save identifier appended to beginning of record, while its data is appended to the end of the record.
sv id		SURFACE (1) class	derived from SURFACE class
ctrl	else		it is a subtype reference; save identifier is followed immediately by its data, both enclosed by subtype_start and subtype_end.
sv id		surface (2) class	derived from surface class

Description: Used to determine which class specified the cone. A subtype reference is inline with a definition and is surrounded by curly braces { }, or Tag 15 and 16.

SURFACE (1) class

Purpose: Defines a generic surface as an object in the model.

Derivation: SURFACE : ENTITY : –

Data Elements:

prim	No data	This class does not save any data

Description: A SURFACE provides the basic framework for the range of surface geometries implemented in the modeler. It may be referenced by more than one entity.

surface (2) class

Purpose: Base class for all ACIS surface types that defines the basic virtual functions that are supplied for all specific surface classes.

Derivation: surface :–

Data Elements:	ctrl	if_cond	if save_version_number is greater than or equal to BNDSUR_VERSION
	prim	interval	*u* range
	prim	interval	*v* range

Description: The surface class is the base class that all ACIS surface types (plane, cone, sphere, torus, and spline) are derived. The surface class defines the basic virtual functions that are supplied for all specific surface classes. Some of these functions are pure; i.e., the derived classes must define their own version; others have default definitions that can be used by the derived classes.

surface type

Purpose: More detailed definition of surface.

Derivation: surface :–

Data Elements:	prim	ident	Surface type
	ctrl	if_cond	if Surface type is set to “null_surface” No surface saved
	ctrl	if_cond	if Surface type is set to “plane”
	sv id	plane (2) class	Plane definition
	ctrl	if_cond	if Surface type is set to “cone”
	sv id	cone (2) class	Cone definition
	ctrl	if_cond	if Surface type is set to “sphere”
	sv id	sphere (2) class	Sphere definition
	ctrl	if_cond	if Surface type is set to “torus”
	sv id	torus (2) class	Torus definition
	ctrl	if_cond	if Surface type is set to “spline”
	sv id	spline (2) class	Spline definition
	ctrl	if_cond	if Surface type is set to “stripc”
	sv id	stripc (2) class	Strip curve definition

Description: Refer to purpose.

“surfcur”

Purpose: Represents a spline curve projected onto a surface within the given fit tolerance.

Derivation: surf_int_cur : int_cur : subtrans_object : subtype_object : –

Data Elements:

prim	subtype_start	Left curly braces, “{” or Tag 15
prim	write sv id	save identifier for this particular subtype
ctrl	if_cond	if save_version_number is less than INTCURVE_VERSION
sv id	bs3_curve_def	
prim	real	fit tolerance
prim	newline	
sv id	surface (2) class	surface 1
prim	newline	
sv id	surface (2) class	surface 2
prim	newline	
sv id	bs2_curve_def	surface 1
prim	newline	
sv id	bs2_curve_def	surface 2
ctrl	else if_cond	if save_version_number is less than PARCUR_VERSION
sv id	bs3_curve_def	
prim	real	fit tolerance
prim	newline	
sv id	surface data	surface 2
prim	newline	
sv id	surface (2) class	surface 1
prim	newline	
sv id	bs2_curve_def	surface 2
prim	newline	
sv id	bs2_curve_def	surface 1
ctrl	else	
sv id	int_cur	Generic int_cur data
prim	logical	Consistent version: “surf2” or “surf1”
prim	subtype_end	Right curly braces, “}” or Tag 16

Description: Refer to the Purpose.

“surfintcur”

Purpose: Represents the spline curves obtained from the intersection of two surfaces.

Derivation: int_int_cur : int_cur : subtrans_object : subtype_object : –

Data Elements:

prim	subtype_start	Left curly braces, “{” or Tag 15
prim	write sv id	save identifier for this particular subtype
sv id	int_cur	Generic int_cur data
prim	subtype_end	Right curly braces, “}” or Tag 16

Description: This class represents the spline curves obtained from the intersection of two surfaces. The given surfaces must not be tangential, except at the ends of the parameter range.

“SURFNORM”

Purpose: Composes a law mathematic function that returns the normal to a surface at a given position.

Derivation: surfnorm_law : unary_law : law : –

Data Elements:

prim	string	The word “SURFNORM” followed by something in parenthesis appears somewhere within this double quoted string.

Description: surf returns the positions of the defining surface at the parameter value. In other words, 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. This law does not normalize the returned vector, because many applications only require the direction of the vector and not its normalized value.

“SURFPERP”

Purpose: Composes a law mathematic function that returns the position on a surface of point projected perpendicular to surface.

Derivation: surfperp_law : multiple_data_law : law : –

Data Elements:			
	prim	string	The word “SURFPERP” followed by something in parenthesis appears somewhere within this double quoted string.

Description: surfperp returns the *uv* position on the given surface, my_surface_law_data, that is closest to the position given by my_position_law. The optional argument my_uv_guess_law specifies a first guess by the user at the correct answer, which may speed up the calculation.

“SURFVEC”

Purpose: Composes a law mathematic function that returns a parameter vector on a surface.

Derivation: surfvec_law : multiple_law : law : –

Data Elements:			
	prim	string	The word “SURFVEC” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The surfvec returns a parameter vector on my_surflaw at my_paralaw that is tangent to my_veclaw. It also returns a new parameter value if the input parameter value is on a singularity.

For example, if my_surflaw is a sphere and the my_paralaw is at the North pole, then this law returns the parameter vector (–1, 0) and the parameter position (pi/2, *v*), where *v* indicates the direction my_veclaw is pointing in. Hence, surfvec returns an array of four values: the first two are the parameter vector, and the second two are the potentially new parameter position. The parameter position, except in the case of singularities, equals my_paralaw.

“sweepsur”

Purpose: Defines the perpendicular sweep of a planar profile curve along a path curve.

Derivation: sweep_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:

prim	subtype_start	Left curly braces, “{” or Tag 15
prim	write sv id	save identifier for this particular subtype
ctrl	if_cond	if save_version_number is less than SPLINE_VERSION or save_version_number is less than LAW_VERSION, system warning with LAW_SAVE_APPROX.
ctrl	else	
prim	logical	Write either “angled” or “normal” for the path normal.
prim	newline	
sv id	curve type	Shape curve
prim	newline	
sv id	curve type	Path curve
prim	newline	
prim	logical	Write either “angled” or “normal” for the sweep normal.
prim	newline	
prim	vector	Write the vector for the shape normal.
prim	newline	
prim	position	Write the position for the path start.
prim	newline	
prim	vector	Write the vector for the start frame row(0).
prim	newline	
prim	vector	Write the vector for the start frame row(1).
prim	newline	
prim	vector	Write the vector for the start frame row(2).
prim	newline	
prim	real	Start *u*-parameter

prim	real	End *u*-parameter
prim	real	Start *v*-parameter
prim	real	End *v*-parameter
prim	newline	
ctrl	if_cond	if save_version_number is greater than or equal to LAW_VERSION
sv id	law	Write out the rail law.
sv id	law	Write out the draft law.
sv id	law	Write out the scale law.
sv id	spl_sur	Generic spl_sur data
prim	subtype_end	Right curly braces, “}” or Tag 16

Description: This class defines the perpendicular sweep of a planar profile curve along a path curve. The start of the path is in the plane of the shape curve.

“swepttapersur”

Purpose: Class to describe a swept–tapered surface, in which a surface is tapered about an edge by a constant angle relative to a draft angle.

Derivation: swept_tpr_spl_sur : edge_tpr_spl_sur : taper_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:

prim	subtype_start	Left curly braces, “{” or Tag 15
prim	write sv id	save identifier for this particular subtype
sv id	“edgetapersur”	Save the information from the edge_tpr_spl_sur.
prim	real	sine angle
prim	real	cosine angle
prim	subtype_end	Right curly braces, “}” or Tag 16

Description: Class to describe a swept–tapered surface, in which a surface is tapered about an edge by a constant angle relative to a draft angle. The surface is a ruled surface between two u parameter curves.

“sys”

Purpose: Defines a base class for the Kernel Component.

Derivation: ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: The base class from which all attributes defined in the Kernel Component are derived.

“T”

Purpose: Composes a law function that uses the identity law to take and return the first input argument.

Derivation: identity_law : law : –

Data Elements: prim string The character “T” not already part of another word appears somewhere within this double quoted string.

Description: Most law functions accept numbers as input arguments. This is accomplished using the identity laws. a1, x, u, and t are the same; a2, y, and v are the same; and a3 and z are the same.

When the identity is used as input into a law function, it is sometimes followed by an integer *n* that specifies its index into the input argument list.

A law expression with a1 and law1 followed by a number and a law is invalid, because each is requesting a different argument type as the first element of the argument list. To correct this problem, specify the ordering of the arguments in the input argument list (e.g., number and then law) and then specify the index number (e.g., x and law2, or e.g., a1 and law2).

“t3”

Purpose: Implements planar triangular elements.

Derivation: TRI3_ELEM : ELEM2D : ELEM : ENTITY : –

Data Elements: sv id “2d” Save parent class ELEM2D

Description: Contains all of the information for each triangular patch on the mesh surface.

"TAN"

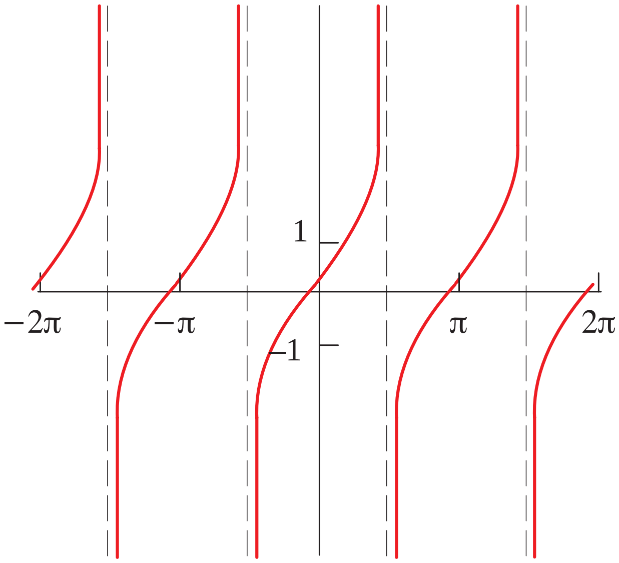
Purpose: Composes a law mathematic function that finds the tangent.

Derivation: tan_law : unary_law : law : –

Data Elements:	prim	string	The word "TAN" followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$$y = \tan x = \frac{\sin x}{\cos x}$$



"tan_xedge"

Purpose: Attaches an attribute to the cross edges of the blend sheet.

Derivation: ATTRIB_TAN_XEDGE : ATTRIB_BLINFO : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: Attribute that attaches to cross edges of the blend sheet, that meet the blend body tangentially. These occur when the sheet face has been made for a blended cuspate vertex. The unblended edge records the curve where the sheet edge lies. This is used internally during blend1, but should be removed from the sheet at the end of this stage.

“TANH”

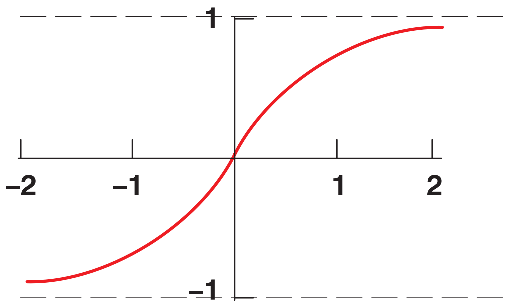
Purpose: Composes a law mathematic function that finds the hyperbolic tangent.

Derivation: tanh_law : unary_law : law : –

Data Elements:	prim	string	The word “TANH” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$$y = \tanh x = \frac{\sinh x}{\cosh x} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



“tapersil”

Purpose: Creates an interpolated curve subtype which can precisely represent a silhouette curve formed by applying a taper.

Derivation: taper_silh_int_cur : int_cur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start	Left curly braces, “{” or Tag 15
	prim	write sv id	save identifier for this particular subtype
	sv id	int_cur	Save the interpolated curve data
	prim	vector	direction of curve
	prim	real	taper angle
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: Refer to Purpose.

“tapersur”

Purpose: Creates an edge–tapered surface.

Derivation: taper_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start	Left curly braces, “{” or Tag 15
	prim	write sv id	save identifier for this particular subtype
	sv id	spl_sur	save specific spline surface data
	sv id	int_cur	save specific interpolated curve data
	ctrl	if_cond	if save_version_number is less than TAPER_VERSION
	prim	vector	direction of taper
	prim	real	sine of angle
	prim	real	cosine of angle
	prim	interval	*u* range
	prim	interval	*v* range
	prim	integer	closed in *u* either “open”, “closed”, “periodic”, or “unknown”.
	ctrl	if_cond	if save_version_number is greater than or equal to DISCONTINUITY_VERSION
	prim	newline	
	prim	discontinuity_info	U Parameter values of discontinuities
	prim	newline	
	prim	discontinuity_info	V Parameter values of discontinuities
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: Class to describe an edge–tapered surface, in which a surface is tapered by a constant angle, relative to a draft angle, about an edge. The surface is a ruled surface between two u parameter curves.

“TERM”

Purpose: Composes a law mathematic function that returns a single term from a given multi–dimensional function.

Derivation: term_law : multiple_law : law : –

Data Elements:	prim	string	The word “TERM” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The term law symbol returns a single dimensional element (coordinate) of a multidimensional (my_law1) function. my_term is an integer greater than zero (0) that specifies which element to grab. This is useful if my_law1 is a curve in *x*, *y*, *z* space, and one of the coordinates needs to be isolated.

In other words, assume my_law is a vector field defined by “vec(x, x+1, x+2, x+3)”. A declaration like (law:eval ”term(my_law, 4)” 1) evaluates the fourth coordinate of my_law, x+3, at the value 1. It returns 4. A declaration like (law:eval ”term(my_law, 3)” 1) evaluates the third coordinate of my_law, x+2, at the value 1. It returns 3.

The next example first creates an edge, called my_edge. Then it creates a law, called my_law, that is the composition of three laws. The “map” law symbol maps that parameter domain of my_edge or “edge1” to the closed interval [0,1]. The “term” law symbol returns the “x” coordinate of the “cur” function that returns the position of the curve “edge1”. Next my_maxpoint is defined as the numerical minimum of the law my_law over the domain [0.5,1]. Then my_testcur is evaluated at my_maxpoint, and the result is plotted. The plotted point represents the point on the curve that has the lowest *x* coordinate.

“text_ent”

Purpose: Routine to restore a TEXT_ENT entity from file.

Derivation: TEXT_ENT : ENTITY : –

Data Elements:	prim	position	Location of baseline at start of first character.
	prim	string	String to be displayed.
	prim	string	Font style used to display string. (NULL means to use the current font).
	prim	integer	Size in points of font used to display string.

Description: These items define the font and the size of the text, as well as the text itself and its location.

times

Purpose: Composes a law mathematic function that uses the times or multiplication (“*”) operator.

Derivation: times_law : binary_law : law : –

Data Elements:	prim	string	The character “*” not already part of another word appears somewhere within this double quoted string and has elements preceding it and following it.

Description: Parsing actually involves the “*” character. my_law1 and my_law2 can be any valid law mathematic function. Both my_law1 and my_law2 can be multiple dimensions; the smaller of the two is padded with zeros.

“torus”

Purpose: Identifier used by more than one class.

Derivation: None

Data Elements:	ctrl	if_cond		if not a subtype reference; save identifier appended to beginning of record, while its data is appended to the end of the record.
	sv id		TORUS (1) class	derived from TORUS class
	ctrl	else		it is a subtype reference; save identifier is followed immediately by its data, both enclosed by subtype_start and subtype_end.
	sv id		torus (2) class	derived from torus class

Description: Used to determine which class specified the torus. A subtype reference is inline with a definition and is surrounded by curly braces { }, or Tag 15 and 16.

TORUS (1) class

Purpose: Defines a torus as an object in the model.

Derivation: TORUS : SURFACE : ENTITY : –

Data Elements:	sv id	torus (2) class	Torus definition

Description: A TORUS is a circular thickening of a circular spine, defined by a center, normal, major radius, and minor radius.

The minor radius is the radius of the thickening circle. The major radius is the radius of the spine from its center point. The normal determines the orientation in space of the torus.

torus (2) class

Purpose: Represents tori.

Derivation: torus : surface : –

Data Elements:	ctrl	if_cond			if used as a subtype reference
	prim		subtype_start		Left curly braces, “{” or Tag 15
	ctrl		if_cond		if save_version_number is less than the SURFACE_VERSION
	prim			integer	torus type
	ctrl		else		
	prim			string	save identifier for torus “torus”
	prim	position			center
	prim	vector			normal
	prim	real			major radius
	prim	real			minor radius
	ctrl	if_cond			if save_version_number is greater than or equal to SURFACE_VERSION
	prim		vector		*uv* origin direction
	prim		logical		“forward_v” or “reverse_v”
	sv id	surface (2) class			information about surface; refer to another section of this manual.
	ctrl	if_cond			if used as a subtype reference
	prim		subtype_end		Right curly braces, “}” or Tag 16

Description: A torus is defined by a circular spine and a circular cross-section at each point on the spine. The spine of a torus is defined by a center point, normal, and major radius. The circular cross-section is defined by a minor radius.

A normal torus (donut) is defined when the major radius is larger than the minor radius. Special degenerate cases (lemon, vortex, and apple) occur if the major radius is smaller than or equal to the minor radius. Two data members define the parameterization of the torus:

unit_vector uv_oridir
Direction from the center of the torus to the origin of parameter space.

logical reverse_v . . .
Constant *u*-parameter lines are circles around the torus axis, normally clockwise, but reversed if this is TRUE.

The *u*-parameter is the latitude, with zero on the circle of greatest radius about the torus axis, and the positive direction in the direction of the torus axis. The *u*-parameter range depends upon the relative values of the major and minor radii. For a doughnut, where the major radius is greater than the magnitude of the minor, it runs from –pi to pi, and is periodic. For degenerate tori, where the magnitude of the major axis is less than that of the minor, it runs from -*U* to *U*, where *U = arccos(–maj / |min|)*, and the surface is singular at each end of the range. The *v*-parameter is the longitude, running from –pi to pi, with 0 on the meridian containing uv_oridir, and increasing in a clockwise direction around the torus axis, unless reverse_v is TRUE, when it increases in an counterclockwise direction.

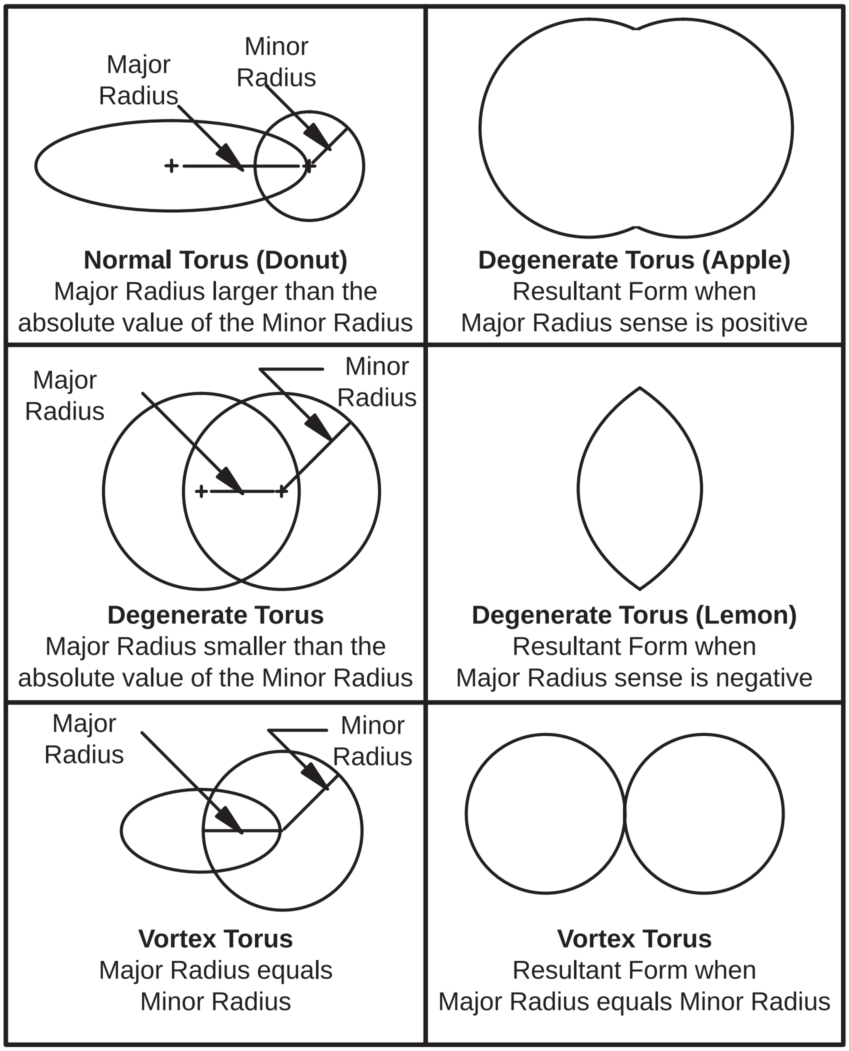
Let *N* be normal and *Q* uv_oridir, and let *R* be *N* X *Q*, negated if reverse_v is true. Let *r* be the absolute value of the minor radius. Then:

```
pos = center + r* sinu* N +
(major_radius + (r* cosu)) * (cosv* Q + sinv R)
```

This parameterization is left-handed for a convex torus and right-handed for a hollow one, if reverse_v is false, and reversed if it is TRUE. When the torus is transformed, the sense of reverse_v is inverted if the transform includes a reflection. No special action is required for a negation.

In summary:

– Tori are not true parametric surfaces.
– Tori are closed in *v* but may or may not be closed in *u*.
– Degenerate tori are not periodic in *u*; nondegenerate tori are periodic in *u* (–pi to pi with period *2*pi).
– All tori are periodic in *v* (with range –pi to pi, and period *2*pi).
– Degenerate tori are singular in *u* at the poles (apices); all other values of *u* and *v* are non-singular.



Normal Torus (Donut)
Major Radius larger than the
absolute value of the Minor Radius

Degenerate Torus (Apple)
Resultant Form when
Major Radius sense is positive

Degenerate Torus
Major Radius smaller than the
absolute value of the Minor Radius

Degenerate Torus (Lemon)
Resultant Form when
Major Radius sense is negative

Vortex Torus
Major Radius equals
Minor Radius

Vortex Torus
Resultant Form when
Major Radius equals Minor Radius

Figure 7-1. torus Class Definition

“TRANS”

Purpose: Composes a law mathematic function that transforms positions.

Derivation: transform_law : multiple_data_law : law : –

Data Elements: prim string The word “TRANS” followed by something in parenthesis appears somewhere within this double quoted string: **`trans`** (my_law, my_transf_law_data)

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.

“TRANS#”

Purpose: Composes a law function with a tag for a transform used as an input argument.

Derivation: transform_law_data : law_data : –

Data Elements:

prim	string			The word “TRANS” followed by an integer appears somewhere within this double quoted string.
prim	integer			An integer greater than 0. It indicates how many law data support items there are.
ctrl	repeat			Repeat the next steps for each law data support item.
prim		string		This is a double quoted string with one of the words: “TRANS”, “EDGE”, “SURF”, or “WIRE”.
ctrl		if_cond		If the string is the double quoted ”TRANS”
sv id			“transform”	Save the underlying transform information or a pointer to it.

Description: Some law functions, such as rotate and trans, accept transforms as input arguments.

When a transform is used as input into a law function, it is always followed by an integer *n* that specifies its index into the input argument list. The index numbering starts at 1. For any given index number *n*, the argument list has to contain at least *n* arguments.

A law expression with trans1 and law1 followed by a transform and a law is invalid, because each is requesting a different argument type as the first element of the argument list. To correct this problem, specify the ordering of the arguments in the input argument list (e.g., law and then transform) and then specify the index number (e.g., trans2 and law1).

If the law to which a trans# is passed returns a vector, the law to use is rotate. If the law to which a trans# is passed returns a position, the law to use is trans.

“transform”

Purpose: Represents an overall transformation applied to a BODY.

Derivation: TRANSFORM : ENTITY : –

Data Elements: prim transform Transformation matrix

Description: The TRANSFORM class represents an overall transformation applied to a BODY. TRANSFORM allows object-space transformations to be applied without the need to recompute the BODY geometry.

It allows a general affine transformation, but records the separate elements of the transformation (scaling, rotation, translation, etc.) to simplify the task of geometry transformation in the common case of solid-body transformations.

“tri3sur”

Purpose: Represents a mesh surface consisting only of planar triangular elements.

Derivation: tri3_msh_sur : msh_sur : –

Data Elements: prim No data This class does not save any data

Description: Depicts a mesh surface consisting only of planar triangular elements and is derived from the abstract msh_sur class.

“tsl”

Purpose: Defines a base class for a specific application developer.

Derivation: ATTRIB_TSL : ATTRIB : ENTITY : –

Data Elements: prim No data This class does not save any data

Description: Identifier used externally to identify an particular entity type. This is only used within the save/restore system for translating to/from external file format, but must be unique amongst attributes derived directly from ATTRIB, across all application developers.

“tubesur”

Purpose: A surface that is the envelope of a fixed-radius circle.

Derivation: tube_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start	Left curly braces, “{” or Tag 15
	prim	write sv id	save identifier for this particular subtype
	sv id	spl_sur	spline surface
	prim	real	Radius
	prim	newline	
	sv id	curve type	
			spine curve
	prim	newline	
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: This class represents a surface that is the envelope of a fixed-radius circle centered on a point on a given curve, and normal to the curve at each point. This has been replaced by the new pipe_spl_sur.

“TWIST”

Purpose: Composes a law mathematic function that returns a twisted vector field about a given path.

Derivation: twist_path_law : multiple_law : law : –

Data Elements:	prim	string	The word “TWIST” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The twist law mathematic function takes in one value and returns a vector that is formed by rotating my_vector_field about my_path_law by an angle (in radians) given by my_twist_law. my_vector_field is a law that takes in one value and returns a vector. my_path_law is a law that takes in one value and returns a position. my_twist_law is a law that takes one value and returns one value. This is used for creating rail laws for sweeping with twist.

“two_ends”

Purpose: Represents data for a blend using a variable radius at both ends.

Derivation: var_rad_two_ends : var_radius :–

Data Elements:	prim	real	Start radius
	prim	real	End radius

Description: Refer to the Purpose.

“U”

Purpose: Composes a law function that uses the identity law to take and return the first input argument.

Derivation: identity_law : law : –

Data Elements:	prim	string	The character “U” not already part of another word appears somewhere within this double quoted string.

Description: Most law functions accept numbers as input arguments. This is accomplished using the identity laws. a1, x, u, and t are the same; a2, y, and v are the same; and a3 and z are the same.

When the identity is used as input into a law function, it is sometimes followed by an integer *n* that specifies its index into the input argument list.

A law expression with a1 and law1 followed by a number and a law is invalid, because each is requesting a different argument type as the first element of the argument list. To correct this problem, specify the ordering of the arguments in the input argument list (e.g., number and then law) and then specify the index number (e.g., x and law2, or e.g., a1 and law2).

“UNBEND”

Purpose: Creates a law to unbend from a position around an axis in a given direction a specified amount.

Derivation: unbend_law : multiple_law : law : –

Data Elements:	prim	string	The word “UNBEND” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The variables to this law function are laws. However, my_pos, my_axis, and my_direction have to return three elements [i.e., VEC(0, 0, 0)], while my_distance has to return one element.

“undefc”

Purpose: Identifier used by more than one class.

Derivation: None

Data Elements:	ctrl	if_cond		if not a subtype reference; save identifier appended to beginning of record, while its data is appended to the end of the record.
	sv id		UNDEFC (1) class	derived from UNDEFC class
	ctrl	else		it is a subtype reference; save identifier is followed immediately by its data, both enclosed by subtype_start and subtype_end.
	sv id		undefc (2) class	derived from undefc class

Description: Used to determine which class specified the undefc. A subtype reference is inline with a definition and is surrounded by curly braces { }, or Tag 15 and 16.

UNDEFC (1) class

Purpose: Defines a curve that is undefined except for its end points.

Derivation: UNDEFC : CURVE : ENTITY : –

Data Elements:	sv id	undefc (2) class	Curve definition

Description: This class defines an undefined curve that records the start and end points, directions, and curvatures. The start point has a parameter value of 0 and the end point has a parameter value of 1. No other points are defined.

undefc (2) class

Purpose: Denotes a curve that is undefined except for its end points, for which there are explicit positions, directions, and curvatures.

Derivation: undefc : curve : –

Data Elements:

ctrl	if_cond		if used as a subtype reference
prim	subtype_start		Left curly braces, “{” or Tag 15
ctrl	if_cond		if save_version_number is less than the CURVE_VERSION
prim		integer	straight type
ctrl	else		
prim		string	save identifier; “undefc”
prim	position		Start point
prim	vector		Start direction
prim	vector		Start curvature
prim	position		End point
prim	vector		End direction
prim	vector		End curvature
sv id	curve (2) class		Generic curve data
ctrl	if_cond		if used as a subtype reference
prim	subtype_end		Right curly braces, “}” or Tag 16

Description: This class denotes a curve that is undefined except for its end points, for which there are explicit positions, directions, and curvatures. This class is used in blending to allow the blend surface to spread at its ends. It may be useful elsewhere, as well.

The curve is parameterized so that the start point has parameter 0, and the end point has parameter 1. No other point lies on the curve so the parameter value is meaningless, but it returns as 0.5.

“units”

Purpose: Specifies the units a model is defined in.

Derivation: UNITS_SCALE : ENTITY : –

Data Elements:

prim	real	Model scale
prim	real	Input scale
prim	real	Output scale

Description: Implements the UNITS_SCALE class. A UNITS_SCALE ENTITY is used to specify what units a model is defined in. It contains a scale factor which specifies the conversion factor between model units and millimeters.

"unknown"

Purpose: Represents common data and functionality that is mandatory in all classes that are permanent objects in the model.

Derivation: ENTITY : –

Data Elements:	sv id	ENTITY	used with the ENTITY class

Description: Refer to purpose.

"V"

Purpose: Composes a law function that uses the identity law to take and return the second input argument.

Derivation: identity_law : law : –

Data Elements:	prim	string	The character "V" not already part of another word appears somewhere within this double quoted string.

Description: Most law functions accept numbers as input arguments. a1, x, u, and t are the same; a2, v, and y are the same; and a3 and z are the same.

"varblendsplsur"

Purpose: Implementation of the base class for variable radius and other nonpipe blends. Derived from blend_spl_sur.

Derivation: var_blend_spl_sur : blend_spl_sur : spl_sur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start	Left curly braces, “{” or Tag 15
	prim	write sv id	save identifier for this particular subtype
	ctrl	if_cond	if rb_envelope and save_version_number is less than BL_ENV_SF_VERSION, system error with BAD_SAVE_FORMAT
	sv id	blend_spl_sur	Generic blend surface data
	sv id	curve type	Slicing plane curve
	prim	newline	
	prim	logical	Either “concave” or “convex”
	ctrl	if_cond	if save_version_number is greater than or equal to BL_ENV_SF_VERSION
	prim	logical	Either “rb_snapshot” or ”rb_envelope”
	prim	newline	
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: Implementation of the base class for variable radius and other nonpipe blends. Derived from blend_spl_sur. This class name does not appear in the save file, but is a base class for other subtype identifiers that do appear in the save file. The flag “rb_snapshot” or “rb_envelope” indicates which evaluator method was used.

var_cross_section

Purpose: Represents the cross section of a blend surface.

Derivation: var_cross_section : –

Data Elements:	ctrl	if_cond			if save_version_number is less than CONSISTENT_VERSION
	prim		integer		form_data
	ctrl	else			
	enum		cross_section_forms		Form
	ctrl		if_cond		if cross section form is set to thumbweights
	prim			real	Left thumbweight
	prim			real	Right thumbweight
	ctrl		else if_cond		if cross section form is set to round chamfer
	prim			logical	“no_radius” or “radius”
	ctrl	if_cond			if height saved
	sv id		var_radius		Round height specification

Description: The cross section of a blend surface corresponds to the *u*-parameter of the surface. When evaluating the surface, a slice is taken at the given *v*-parameter, and then that slice is evaluated at the *u*-parameter.

During the initial construction of the blend surface geometry, the cross section is irrelevant. It first comes into play when the blend surface is intersected with other faces.

Only circular cross sections are available in standard blending. The Advanced Blending Component allows other shapes. Parameterization runs from 0 to 1, and that covers the whole section.

var_radius

Purpose: Defines variable radius information for a variable radius blend.

Derivation: var_radius : –

Data Elements:	enum	rad_form_ents	Form
	prim	logical	Either “uncalibrated” or “calibrated”
	prim	real	Radius start parameter
	prim	real	Radius end parameter
	ctrl	if_cond	if radius form is set to two ends
	sv id	var_rad_two_ends	Specific radius data
	ctrl	if_cond	if radius form is set to functional
	sv id	var_rad_functional	Specific radius data
	ctrl	if_cond	if radius form is set to elliptical
	sv id	var_rad_rot_ellipse	Specific radius data
	ctrl	if_cond	if radius form is set to fixed width
	sv id	var_rad_fixed_width	Specific radius data

Description: This class defines a variable radius. Start and end parameters must always be set to something reasonable, even if it’s not calibrated. The parameter range of an edge being blended is reasonable.

“vblend”

Purpose: Defines the vertex blend attribute.

Derivation: ATTRIB_VBLEND : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:	prim	real	Bulge
	ctrl	if_cond	if save_version_number is less than CONSISTENT_VERSION
	prim	integer	Continuity
	ctrl	else	
	enum	bl_continuity	Continuity

Description: Refer to purpose.

“VEC”

Purpose: Composes a law mathematic function that is a vector of arbitrary dimensions.

Derivation: vector_law : multiple_law : law : –

Data Elements:	prim	string	The word “VEC” followed by something in parenthesis appears somewhere within this double quoted string.

Description: This law is a way of combining several sublaws, each of one dimension, into a single law that has several dimensions. All sublaws have to return one dimensional items, although they can have multiple input items.

“vector_attrib”

Purpose: Defines a generic attribute that contains a vector.

Derivation: ATTRIB_GEN_VECTOR : ATTRIB_GEN_NAME : ATTRIB_GENERIC : ATTRIB : ENTITY : –

Data Elements:	prim	vector	Value

Description: Refer to the Purpose.

“vertedge”

Purpose: Contains a list of edge pointers.

Derivation: ATTRIB_VERTEDGE : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:	prim	integer	Number of edges
	ctrl	repeat	Repeat for the number of edges
	prim	$rec_num	Pointer to record in save file for an edge which use vertex

Description: This is used to contain the edge pointer list if there should be more than one pointer. At nonmanifold vertices, there should be a pointer to an edge in each separable manifold region.

“vertex”

Purpose: Represents an end of an EDGE.

Derivation: VERTEX : ENTITY : –

Data Elements:	prim	$rec_num	Pointer to record in save file for an edge which uses vertex
	prim	$rec_num	Pointer to record in save file for point at which vertex lies

Description: A VERTEX embodies the user’s view of a corner of a FACE or the end of an EDGE. It refers to an APOINT in object space and to the groups of EDGEs that it bounds. All EDGEs in each group can be found by following pointers through the COEDGEs.

The VERTEX may contain pointers to multiple EDGEs to provide access to all the EDGEs at the VERTEX, such as when a body is nonmanifold at a VERTEX, or when an unembedded EDGE dangles from a VERTEX of an otherwise well-formed solid.

If all the EDGEs at the VERTEX are WIRE (each adjacent to no FACEs) or if all are embedded (each adjacent to two FACEs and in one manifold group), the VERTEX will contain a pointer to a single EDGE. The others can be found by following the next, previous, and partner pointers of the COEDGEs of the EDGEs as appropriate for WIREs or embedded EDGEs.

“vertexblendsur”

Purpose: Defines the vertex blend surface class.

Derivation: VBL_SURF : spl_sur : subtrans_object : subtype_object : –

Data Elements:	prim	subtype_start	Left curly braces, “{” or Tag 15
	prim	write sv id	save identifier for this particular subtype
	prim	integer	Number of boundaries
	ctrl	repeat	Repeat for the number of boundaries
	sv id	BDY_GEOM	Boundary geometry
	prim	integer	Grid size
	prim	real	Fit tolerance
	prim	subtype_end	Right curly braces, “}” or Tag 16

Description: This class defines the vertex blend surface class. It is defined entirely by the n boundaries that make it up.

“vertex_template”

Purpose: Represents the data to be generated at a facet node.

Derivation: VERTEX_TEMPLATE : ENTITY : –

Data Elements:

sv id	af_node_mapping	Data types to be generated for each vertex

Description: Every node of a facet contains its coordinates. In addition, a node contains a pointer to an array of additional fields. These fields are defined by the vertex template.

“wcs”

Purpose: Defines the WCS class.

Derivation: WCS : ENTITY : –

Data Elements:

prim	transform	Transform to get to model space.

Description: A WCS is used to define a transform which maps input into the coordinate system of the model.

“wire”

Purpose: Represents a collection of EDGEs.

Derivation: WIRE : ENTITY : –

Data Elements:

prim	$rec_num	Pointer to record in save file for next wire in body, shell or subshell
prim	$rec_num	Pointer to record in save file for first coedge in wire
prim	$rec_num	Pointer to record in save file for body or shell containing wire
ctrl	if_cond	if save_version_number is greater than or equal to WIREBOOL_VERSION
prim	$rec_num	Pointer to record in save file for subshell containing wire
prim	logical	(“out” “in”) Containment of wire

Description: A WIRE represents a connected collection of EDGEs, and is owned by a BODY or a SHELL. WIREs stand for construction points and bounded, unbounded or semi-bounded curves. They can represent open or closed profiles, and also general wireframe models that are “unsurfaced”; i.e., have no FACEs.

“WIRE”

Purpose: Composes a law mathematic function that returns the positions of the defining a wire.

Derivation: wire_law : unary_data_law : law : –

Data Elements:	prim	string	The word “WIRE” followed by something in parenthesis appears somewhere within this double quoted string.

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 variance internal to them. All curves other than arcs and lines have non-constant speed.

“WIRE#”

Purpose: Composes a law function with a tag for a wire used as an input argument.

Derivation: wire_law_data : path_law_data : law_data : –

Data Elements:	prim	string	The word "WIRE" followed by an integer appears somewhere within this double quoted string.
	prim	integer	An integer greater than 0. It indicates how many law data support items there are.
	ctrl	repeat	Repeat the next steps for each law data support item.
	prim	string	This is a double quoted string with one of the words: "TRANS", "EDGE", "SURF", or "WIRE".
	ctrl	if_cond	If the string is the double quoted "WIRE"
	prim	integer	Number for how many curves are in the wire.
	ctrl	repeat	For every curve in the wire, repeat the next four steps: take care of the curve type, the starting point, the scale factor, and the interval.
	sv id	curve type	Save the underlying curve for this edge.
	prim	real	Starting point with respect to the curve.
	prim	real	Scale factor to convert to arc-length parameterization.
	prim	interval	Complete interval for the curve, which is its starting and ending points.

Description: Some law functions, such as wire and dwire, accept wires as input arguments.

When a wire is used as input into a law function, it is always followed by an integer *n* that specifies its index into the input argument list. The index numbering starts at 1. For any given index number *n*, the argument list has to contain at least *n* arguments.

A law expression with wire1 and law1 followed by a wire and a law is invalid, because each is requesting a different argument type as the first element of the argument list. To correct this problem, specify the ordering of the arguments in the input argument list (e.g., law and then wire) and then specify the index number (e.g., wire2 and law1).

A wire law is parameterized by the arc length to be equal to the arc length at the end points. Thus, parameterization works to the ends of edges but not to the middle of an edge unless the edge has a constant speed, such as straight lines and circles. The parameter spacing on edges with non-constant speeds is not even.

“X”

Purpose: Composes a law function that uses the identity law to take and return the first input argument.

Derivation: identity_law : law : –

Data Elements:	prim	string	The character “X” not already part of another word appears somewhere within this double quoted string.

Description: Most law functions accept numbers as input arguments. This is accomplished using the identity laws. a1, x, u, and t are the same; a2, y, and v are the same; and a3 and z are the same.

A law expression with a1 and law1 followed by a number and a law is invalid, because each is requesting a different argument type as the first element of the argument list. To correct this problem, specify the ordering of the arguments in the input argument list (e.g., number and then law) and then specify the index number (e.g., x and law2, or e.g., a1 and law2).

“xedge”

Purpose: Attaches to cross edges of the blend sheet, recording the blended edge giving rise to the face on one side of the cross edge.

Derivation: ATTRIB_XEDGE : ATTRIB_BLINFO : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: This class attaches to cross edges of the blend sheet, recording the blended edge giving rise to the face on one side of the cross edge. This is used internally during blend1, but it should be removed from the sheet at the end of this stage.

"xvert"

Purpose: Implements the derived blend attribute for flagging a blend sheet pointed vertex.

Derivation: ATTRIB_XVERT : ATTRIB_BLINFO : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: This class implements the derived blend attribute for flagging a blend sheet pointed vertex.

"xverted"

Purpose: Defines an attribute to be attached to point vertices of the vertex blend sheet.

Derivation: ATTRIB_XVERTED : ATTRIB_BLINFO : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:	prim	No data	This class does not save any data

Description: This class defines an attribute to be attached to point vertices of the vertex blend sheet, pointing to the associated edge that was blended with a zero radius blend.

"Y"

Purpose: Composes a law function that uses the identity law to take and return the second input argument.

Derivation: identity_law : law : –

Data Elements:	prim	string	The character "y" not already part of another word appears somewhere within this double quoted string: **y**, **a2**.

Description: Most law functions accept numbers as input arguments. a1, x, u, and t are the same; a2, v, and y are the same; and a3 and z are the same.

“Z”

Purpose: Composes a law function that uses the identity law to take and return the third input argument.

Derivation: identity_law : law : –

Data Elements:	prim	string	The character “Z” not already part of another word appears somewhere within this double quoted string.

Description: Most law functions accept numbers as input arguments. a1, x, u, and t are the same; a2, v, and y are the same; and a3 and z are the same.