Chapter 5.
Save Identifiers A thru D

This chapter describes the save file information for ACIS and its components.

It also describes the subtype format for the save file. Subtypes and references are frequently used within any given save file, because they reduce the amount of information that has to be stored in the file. Instead of defining information two or more times for items which may already be specified in the file, references are made to the instance of data.

> ***Note:*** *Subtypes may be nested within other subtypes. References to subtypes may be nested within other subtypes.*

ACIS is divided into topology, geometry, and attributes. All classes in ACIS for which save file information is written are derived in some manner from ENTITY. The classes given in this chapter are used in class derivation. Subtypes are derived from one of the three main types: int_cur, par_cur, and spl_sur.

Class Format

The structure of a class definition in the save file is created starting from the inside and then working out. In other words, the save identifier name is saved at the left-most portion of a record, while its data is appended to the end of all other current sv id data elements. The next element in the record in turn saves its save identifier name at the left-most portion of the record while appending its data to the end.

Only save identifier (sv id) names in quotation marks (“ “) are actually written to the save file.

Subtype Format

The structure of a subtype definition in the save file is not created in the same way as a class format. Instead, at the point in the record where a subtype is defined, what is written to the save file is:

subtype_start	left curly braces, "{" or Tag 15
save identifier	subtype name
save data	subtype data
subtype_end	right curly braces, "}" or Tag 16

The information is sequential between the subtype_start and subtype_end and does not obey the inside-out encapsulation of normal save file records.

Law Class Format

The structure of a law definition in the save file is not created in the same way as a class format. Instead, at the point in the record where a law is defined, what is written to the save file is:

5. Double-quotation marks enclosing any mathematically valid combination of law symbols (identifiers). This is the law mathematic function.
6. If there are no law data support elements:
    a. The integer 0 follows the law mathematic function.
    b. This completes the save file information for the law mathematic function. *Stop here.*
7. If there are law data support elements, an integer follows the law mathematic function. It indicates the number of law data support items.
8. Repeat this step for the number of law data support items.
    a. If the next double-quoted string is "EDGE", it is followed by:
        1) The underlying curve (see curve type).
        2) A real for the edge's beginning parameter.
        3) A real for the edge's ending parameter.

b. If the next double-quoted string is “TRANS”, it is followed by the underlying transform data or a pointer to it. (See “transform”).

c. If the next double-quoted string is “SURF”, it is followed by:

   1) The underlying surface (see surface type).

   2) An interval for the *u* space of the surface.

   3) An interval for the *v* space of the surface.

d. If the next double-quoted string is “WIRE”, it is followed by an integer for the number of curves in the wire. Each curve in the wire supplies the following data:

   1) The underlying curve (see curve type).

   2) A real for the starting point with respect to the curve.

   3) A real for the scale factor to convert to arc-length parameterization.

   4) An interval that has starting and ending parameters for curve.

Save File Item Template

<NAME> . . . . . The *Name* title is the save identifier used either in the save file itself, or in this manual to specify a unique class of information. If the identifier could appear in a save file, then it is enclosed within quotation marks (“ ”) in the topic title. Identifiers or identifier strings not in quotation marks do not appear in save files, but are used in this manual to trace the data actually written to a save file.

Save identifier are generally written with all other save identifiers in a record, unless the save data section indicates otherwise (which is the case for subtypes). This means an inside-out methodology is employed: the quoted save identifier appears with other save identifiers at the left in a record, while its data is to the right (or below). The save identifiers within a group are generally separated with dashes (–).

Subtypes, however, write the save identifier immediately following the subtype_start. This is followed immediately by the subtype’s data and subtype_end.

Purpose: The *Purpose* field summarizes the intended purpose of the class save information.

Derivation: The *Derivation* field specifies the derivation of the class. The class described by the template is always listed on the left, followed by its parent, grandparent, etc. The last class in the list is always a base class, indicated by a trailing hyphen (–). Classes with no parents (base classes) show only the class name and a trailing hyphen (–).

Data Element: The *Data Element* field describes the information that is saved for each class. It is divided into three columns.

The left-hand column conveys the type of information that the data is. Types are “prim” for primitives, “ctrl” for controls, “enum” for enumerations, and “sv id” for references to other save items.

The middle column contains the data written by this class to the save file, in the order the data is written.

The right-hand column provides a short description.

Description: The *Description* field describes the class save format, its use, and its surrounding conditions.

“1d”

Purpose: Implements the ELEM1D.

Derivation: ELEM1D : ELEM : ENTITY : –

Data Elements:

sv id	“elem”		parent information
ctrl	if_cond		if type is a straight_type
prim		char	writes an “s” for “straight”
ctrl	else		if type is not a straight_type
sv id		curve type	saves the information for that specific type of curve.

Description: Represents a piece of a composite curve. It contains a curve which forms the piece of the composite one.

“2d”

Purpose: Implements the ELEM2D.

Derivation: ELEM2D : ELEM : ENTITY : –

Data Elements:	sv id	"elem"	save parent information
	ctrl	repeat	perform for node count
	prim	$rec_num	Pointer to records in save file for NODE(s)
	ctrl	repeat	perform for element count
	prim	$rec_num	Pointer to records in save file for ELEMENT(s)

Description: Represents an elemental patch on a mesh surface. It contains the patch boundary information.

"A#"–"Z#"

Purpose: Composes a law function that uses the identity law, x_n, to take and return the *n*-th input argument.

Derivation: identity_law : law : –

Data Elements:	prim	string	The any letter (such as "A") followed by an integer appears somewhere within this double quoted string.

Description: To accept and return the *n*-th input argument into a law function, it is sometimes specified as a_n, where *n* is an integer. The index numbering starts at 1. For any given index number *n*, the argument list has to contain at least *n* arguments.

a1, u, t, and x are the same; a2, v, and y are the same; and a3 and z are the same.

"ABS"

Purpose: Composes a law mathematic function that takes the absolute value of another law mathematic function.

Derivation: abs_law : unary_law : law : –

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

Description: Refer to Purpose statement.

“adjedge”

Purpose: Records blank body edges that are in contact with the sheet boundary, which are called *adjacent edges*.

Derivation: ATTRIB_ADJEDGE : ATTRIB_BLINFO : ATTRIB_SYS : ATTRIB : ENTITY : –

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

Description: This class records blank body edges that are in contact with the sheet boundary, which are called *adjacent edges.* The edge and the parameter value on the edge at the point of contact are recorded. The attribute is attached to the vertex of the sheet in contact with the adjacent edge.

“adjface”

Purpose: Records blank body faces that are in contact with the sheet boundary, which are called *adjacent faces*.

Derivation: ATTRIB_ADJFACE : ATTRIB_BLINFO : ATTRIB_SYS : ATTRIB : ENTITY : –

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

Description: This class records blank body faces that are in contact with the sheet boundary, which are called *adjacent faces.* The face and the parameter value on the face at the point of contact are recorded. The attribute is attached to the vertex of the sheet in contact with the adjacent face.

af_node_mapping

Purpose: Lookup table interface for token types in mesh nodes.

Derivation: af_node_mapping :–

Data Elements:			
	prim	integer	Number of tokens
	ctrl	repeat	Repeat for the number of tokens
	enum	token_name	Token indicating data type to be stored

Description: A node_mapping is a mapping from a set of token types to positions in an array of cells.

"agc"

Purpose: Identifies that the attribute came from the ATTRIB_AGC class.

Derivation: ATTRIB_AGC : ATTRIB : ENTITY : –

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

Description: Identifier used externally to identify a 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.

"AND"

Purpose: Used with PIECEWISE to create a logical AND conditional.

Derivation: and_law : binary_law : law : –

Data Elements: prim string — The word "AND" appears somewhere within this double quoted string.

Description: Refer to Purpose.

"ARCCOS"

Purpose: Composes a law mathematic function that finds the arc cosine.

Derivation: arccos_law : unary_law : law : –

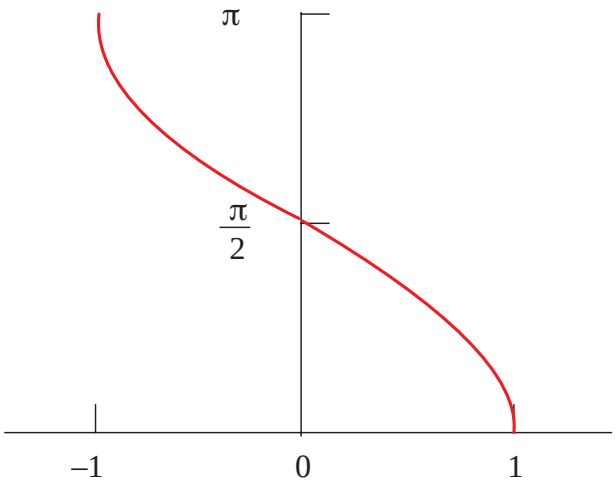
Data Elements: prim string — The word "ARCCOS" followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$y = \arccos x = \cos^{-1} x$

if and only if $\cos y = x$

where $-1 \leqq x \leqq 1$ and $0 \leqq y \leqq \pi$



"ARCCOSH"

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

Derivation: arccosh_law : unary_law : law : –

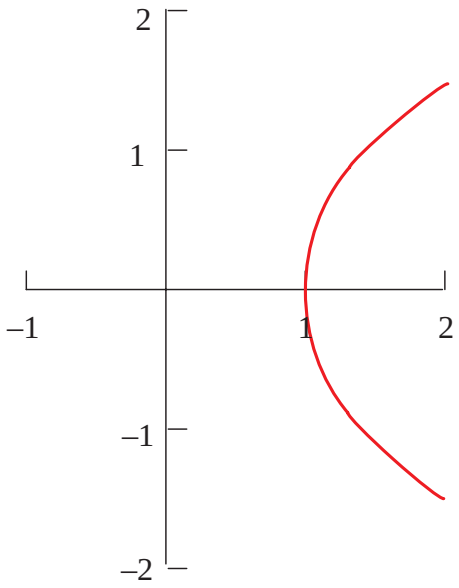
Data Elements:

prim	string	The word "ARCCOSH" followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$y = \text{arccosh}\ x = \cosh^{-1} x$

if $x = \cosh y$ where $x \geqq 1$ and $y \geqq 0$



“ARCCOT”

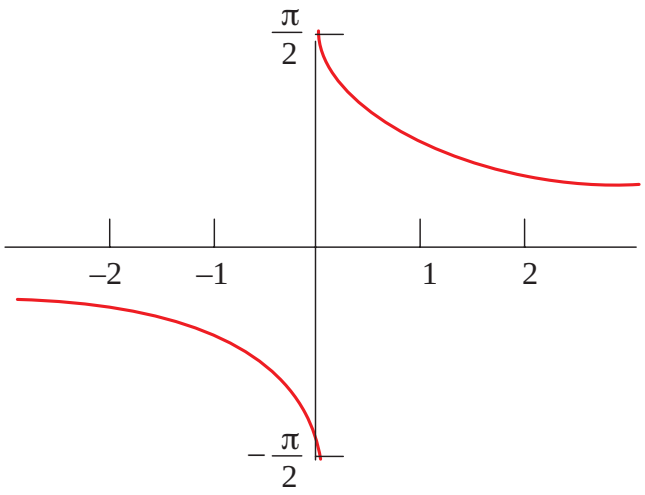
Purpose: Composes a law mathematic function that finds the arc cotangent.

Derivation: arccot_law : unary_law : law : –

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

Description: The mathematical definition is:

$y = \text{arccot } x = \cot^{-1} x$



"ARCCOTH"

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

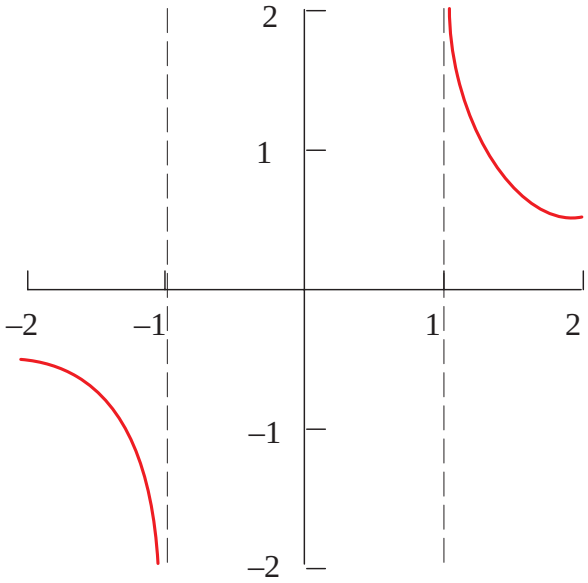
Derivation: arccoth_law : unary_law : law : –

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

Description: The mathematical definition is:

$y = \text{arccoth}\ x = \coth^{-1} x$

if $x = \coth y$ where $|x| > 1$



“ARCCSC”

Purpose: Composes a law mathematic function that finds the arc cosecant.

Derivation: arccsc_law : unary_law : law : –

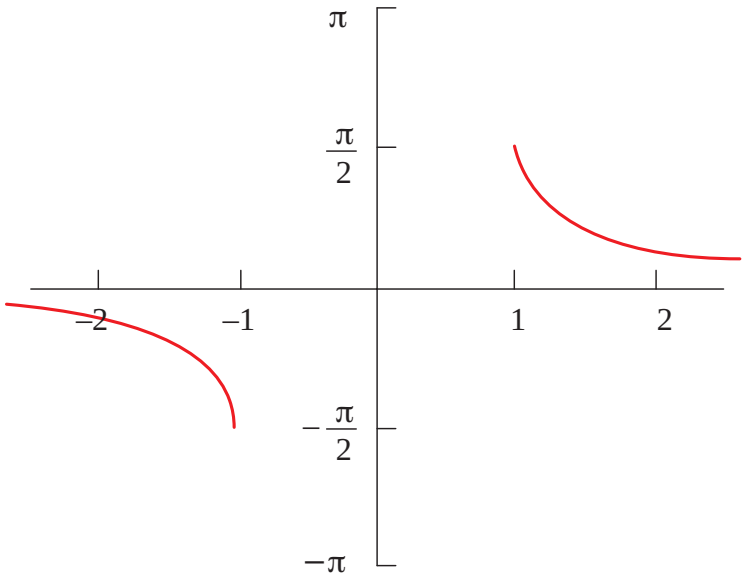
Data Elements: prim string — The word “ARCCSC” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$y = \text{arccsc}\ x = \csc^{-1} x$

if and only if $\csc y = x$

where $|x| \geqq 1$ and $-\frac{\pi}{2} < y < \frac{\pi}{2}$



"ARCCSCH"

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

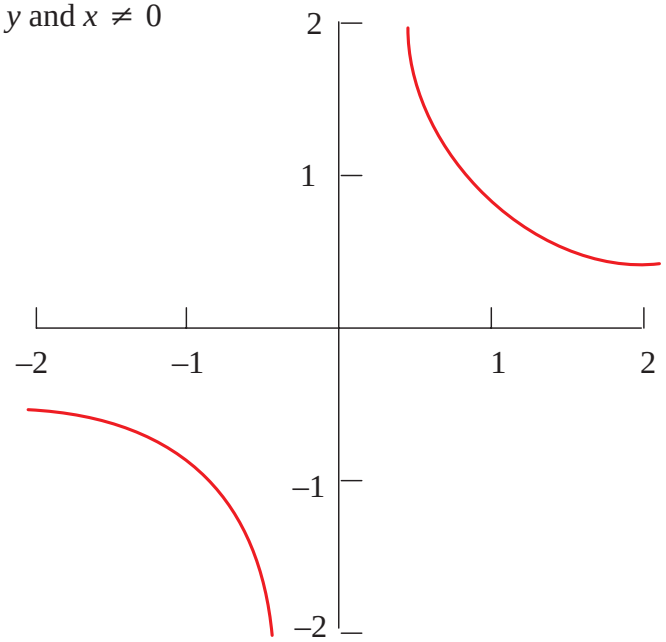
Derivation: arccsch_law : unary_law : law : –

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

Description: The mathematical definition is:

$y = \text{arccsch } x = \text{csch}^{-1} x$

if $x = \text{csch } y$ and $x \neq 0$



“ARCSEC”

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

Derivation: arcsec_law : unary_law : law : –

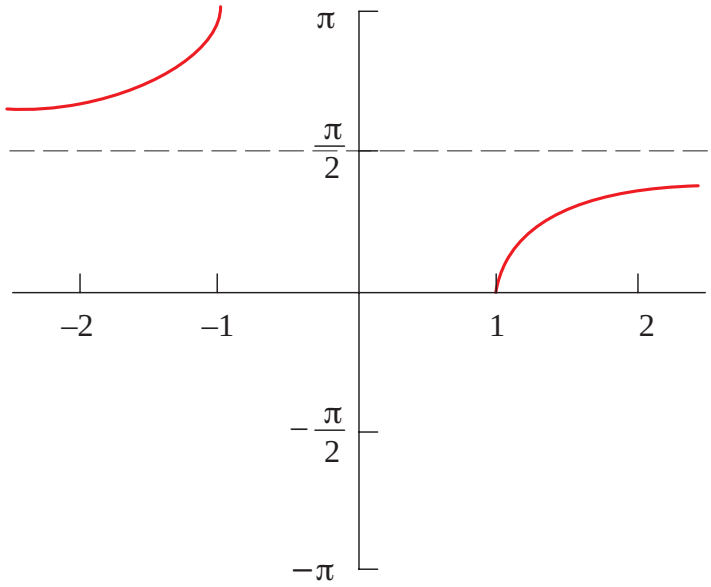
Data Elements: prim string — The word “ARCSEC” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$y = \text{arcsec } x = \sec^{-1} x$

if and only if $\sec y = x$

where $|x| \geqq 1$ and $0 \leqq y \leqq \pi$



"ARCSECH"

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

Derivation: arcsech_law : unary_law : law : –

Data Elements:

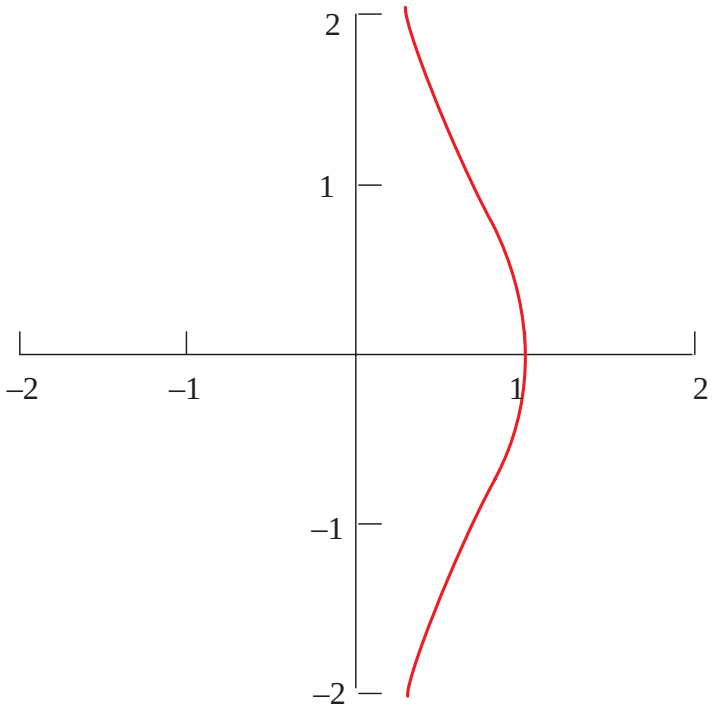
prim	string	The word "ARCSECH" followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$y = \text{arcsech}\ x = \text{sech}^{-1}\ x$

if $x = \text{sech}\ y$

where $0 < x \leqq 1$ and $y \geqq 0$



"ARCSIN"

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

Derivation: arcsin_law : unary_law : law : –

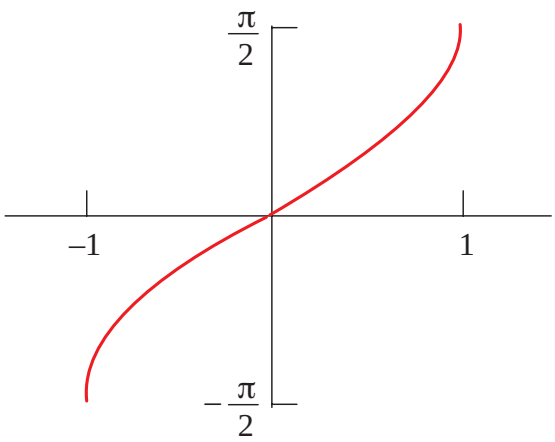
Data Elements: prim string — The word "ARCSIN" followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$y = \arcsin x = \sin^{-1} x$

if and only if $\sin y = x$

where $-1 \leqq x \leqq 1$ and $-\frac{\pi}{2} \leqq y \leqq \frac{\pi}{2}$



"ARCSINH"

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

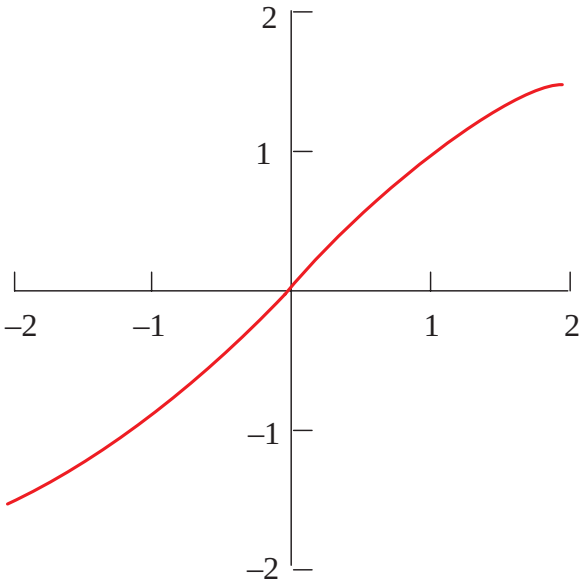
Derivation: arcsinh_law : unary_law : law : –

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

Description: The mathematical definition is:

$y = \text{arcsinh } x = \sinh^{-1} x$

if $x = \sinh y$ for all x



“ARCTAN”

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

Derivation: arctan_law : unary_law : law : –

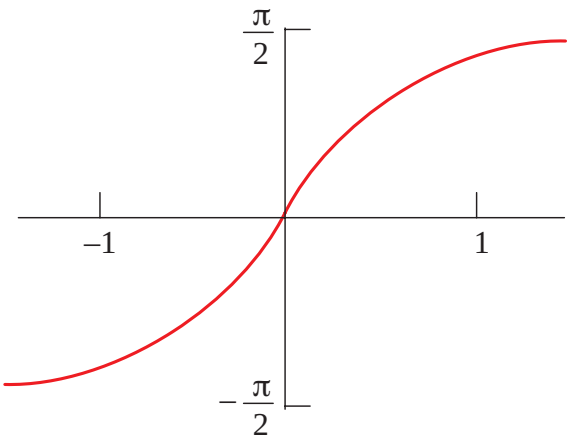
Data Elements: prim string — The word “ARCTAN” followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$y = \arctan x = \tan^{-1} x$

if and only if $\tan y = x$

where $-\frac{\pi}{2} < y < \frac{\pi}{2}$



$y = \arctan x = \tan^{-1} x$
if and only if $\tan y = x$
where –pi/2 < y < pi/2

"ARCTANH"

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

Derivation: arctanh_law : unary_law : law : –

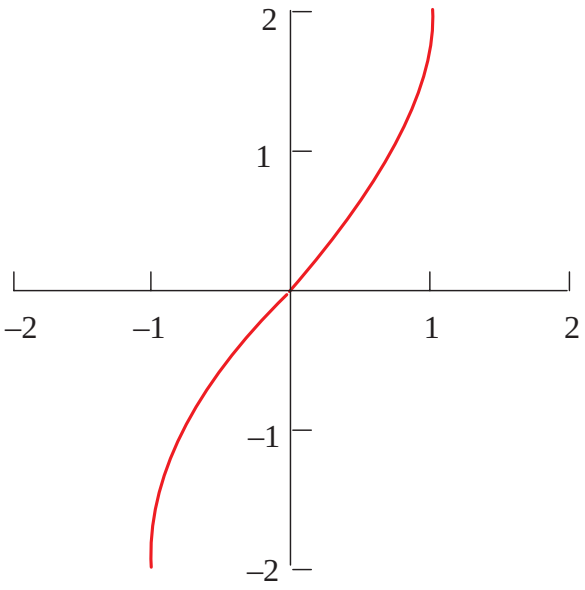
Data Elements: prim string

The word "ARCTANH" followed by something in parenthesis appears somewhere within this double quoted string.

Description: The mathematical definition is:

$y = \text{arctanh}\ x = \tanh^{-1} x$

if $x = \tanh y$ $-1 < x < 1$



"attrib"

Purpose: Represents common data and functionality for all attributes.

Derivation: ATTRIB : ENTITY : –

Data Elements:

prim	$rec_num	Pointer to record in save file for next attribute owned by entity
prim	$rec_num	Pointer to record in save file for previous attribute owned by entity
prim	$rec_num	Pointer to record in save file for owning entity

Description: ATTRIB is the generic class for user and system attributes. It provides housekeeping to maintain ENTITY attribute lists. Every entity may have one or more attributes. These hang off the ENTITY's attribute pointer, and are arranged as a doubly-linked list.

“adv_var_blend”

Purpose: Defines the blend attribute for edge sequence-following blends.

Derivation: ATTRIB_ADV_VAR_BLEND : ATTRIB_VAR_BLEND : ATTRIB_FFBLEND : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:

sv id	var_radius	refer to var_radius data type
ctrl	if_cond	if there are different radii
sv id	var_radius	refer to var_radius data type
sv id	var_cross_section	refer to var_cross_section data type
prim	$rec_num	Pointer to record in save file for left face
prim	$rec_num	Pointer to record in save file for left edge
prim	logical	either “sharp” or “smooth”
prim	$rec_num	Pointer to record in save file for right face
prim	$rec_num	Pointer to record in save file for right edge
prim	logical	“sharp” or “smooth”
prim	$rec_num	Pointer to record in save file for loft edge

Description: This class defines the attribute for edge sequence-following blends in the Advanced Blending Component. It is derived from the ATTRIB_VAR_BLEND class in basic blending.

“attrib_fhl”

Purpose: Faceted Hidden Line Component organizational class.

Derivation: ATTRIB_FHL : ATTRIB : ENTITY : –

Data Elements:

prim	No data	This class does not save any data

Description: ATTRIB_FHL is the organizational class from which the other FHL attribute classes are derived. Its sole purpose is to identify those child classes as belonging to the Faceted Hidden Line Component.

"attrib_fhlgeom"

Purpose: Attaches FHL geometry data to model entities as attributes.

Derivation: ATTRIB_FHL_GEOM : ATTRIB_FHL : ATTRIB : ENTITY : –

Data Elements:

ctrl	if_cond		If the save_version_number is less than the CONSISTENT_VERSION
prim		integer	header for this geometry
prim		integer	next geometry for header
ctrl	else		
prim		$rec_num	Pointer to record in save file for Header for this geometry ATTRIB_FHL_HEAD
prim		$rec_num	Pointer to record in save file for Next geometry for header ATTRIB_FHL_GEOM

Description: All geometric data computed by the Faceted Hidden Line Component is attached to entities in the form of attributes.

The first attribute in the list is an instance of the ATTRIB_FHL_HEAD class. The remaining attributes in the list are instances of the ATTRIB_FHL_GEOM class, and are referred to as geometry attributes. The geometry attributes store the geometry data.

"attrib_fhlhead"

Purpose: Heads the list of FHL attributes attaching hidden line data to the model.

Derivation: ATTRIB_FHL_HEAD : ATTRIB_FHL : ATTRIB : ENTITY : –

Data Elements:

prim	logical		Either "invalid" or "valid"
prim	integer		View token
ctrl	if_cond		If the save_version_number is less than the CONSISTENT_VERSION
prim		integer	Next geometry for this header
ctrl	else		
prim		$rec_num	Pointer to record in save file for Next geometry for this header ATTRIB_FHL_GEOM

Description: All geometric data computed by the Faceted Hidden Line Component is attached to entities in the form of attributes.

The first attribute in the list is an instance of the ATTRIB_FHL_HEAD class and is called the header attribute. The header attribute stores the view-token that identifies which view the attribute set corresponds to.

It also stores a flag that signifies whether or not the attribute set is valid. When an entity undergoes geometric transformations or participates in a Boolean operation, it turns off the validity flag of the header attribute attached to the body. The application program may delete invalid FHL attributes at a later stage.

"attrib_fhlmark"

Purpose: Detects changes to underlying geometry.

Derivation: ATTRIB_FHL_MARK : ATTRIB_FHL_GEOM : ATTRIB_FHL : ATTRIB : ENTITY : –

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

Description: ATTRIB_FHL_MARK is an abstraction of null geometry. Instances of this class are attached to SHELLs or LUMPs and are useful to detect modifications to the entity (such as Boolean operations) that invalidate the FHL attribute set.

"attrib_fhlplist"

Purpose: Attaches FHL polyline data to model ENTITYs.

Derivation: ATTRIB_FHL_PLIST : ATTRIB_FHL_GEOM : ATTRIB_FHL : ATTRIB : ENTITY : –

Data Elements:			
	prim	logical	Visibility: either "invisible" or "visible"
	prim	integer	Number of points
	ctrl	repeat	Repeat for the Number of points
	prim	real	*x*-coordinate
	prim	real	*y*-Coordinate

Description: Line segment data stored in ATTRIB_FHL_SLIST objects is unordered, and so inefficient for plotters. A function converts the unordered segments into polylines and stores them in ATTRIB_FHL_PLIST objects.

ATTRIB_FHL_PLIST objects store the polyline as a list of 2D points. The points in the point list are accessed via an index by some methods as though they are stored in an array.

Visibility information is stored for the polyline, and applies to the entire polyline.

"attrib_fhl_slist"

Purpose: Attaches FHL line segment data to model entities.

Derivation: ATTRIB_FHL_SLIST : ATTRIB_FHL_GEOM : ATTRIB_FHL : ATTRIB : ENTITY : –

Data Elements:

prim	integer		Number of segments
ctrl	repeat		Repeat for the number of segments
prim		real	Start *x*-coordinate
prim		real	Start *y*-coordinate
prim		real	End *x*-coordinate
prim		real	End *y*-coordinate
prim		logical	Either "invisible" or "visible"

Description: ATTRIB_FHL_SLIST stores unordered line segment data in a list of segments. Each segment can be visible or not visible.

ATTRIB_NODE

Purpose: Represents a grazing touch of an intersection curve with the side (boundary) of an element.

Derivation: ATTRIB_NODE : ATTRIB_MESH : ATTRIB : ENTITY : –

Data Elements:

sv id	"ms"	ATTRIB_MESH parent
prim	integer	side
prim	position	node location

Description: Attribute to represent a grazing touch of an intersection curve with the side (boundary) of an element. Used in mesh surface remeshing. We make this attribute "savable" since remeshing is optionally on demand.

ATTRIB_PHL

Purpose: Defines the phl tag identifier.

Derivation: ATTRIB_PHL : ENTITY_PHL : ENTITY : –

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

Description: The phl tag is used by two different classes.

“attrib_var_blend”

Purpose: Defines the blend attribute for variable radius blends.

Derivation: ATTRIB_VAR_BLEND : ATTRIB_FFBLEND : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:			
	prim	$rec_num	Pointer to record in save file for defining curve
	prim	real	*v*-range start
	prim	real	*v*-range end
	prim	logical	Calibrated two ends: “uncalibrated” or “calibrated”
	ctrl	if_cond	If calibrated two ends
	prim	real	Parameter of start radius
	prim	real	Parameter of end radius
	ctrl	if_cond	If the save_version_number is less than the CONSISTENT_VERSION
	prim	integer	radius form
	ctrl	else	
	enum	rad_form_ents	Radius specification format enumeration
	ctrl	if_cond	if the radius form is set for two ends
	prim	real	Start radius
	prim	real	End radius
	ctrl	else if_cond	else if the radius form is set for functional
	prim	logical	Two radii: “one_radius” or “two radii”

sv id		bs2_curve_def	Radius function
ctrl		if_cond	if there are two radii
sv id		bs2_curve_def	Right radius function
ctrl	else if_cond		else if the radius form is set for fixed width
prim		real	Width
ctrl	if_cond		if the save_version_number is less than the CONSISTENT_VERSION
prim		integer	section form
ctrl	else		
enum		sec_form_ents	Cross section form enumeration
ctrl	if_cond		if the section form is set for thumbweights
prim		real	Left thumbweight
prim		real	Right thumbweight
ctrl	if_cond		if the save_version_number greater than or equal to SAFERANGE_VERSION)
prim		logical	sense data, direction of blend with respect to the defining curve: "forward" or "reversed"
prim		$rec_num	Pointer to record in save file for first edge of blend sequence
prim		$rec_num	Pointer to record in save file for last edge of blend sequence

Description: Refer to the Purpose.

BDY_GEOM

Purpose: The base class for different representations of the boundary.

Derivation: BDY_GEOM : –

Data Elements:

ctrl	if_cond		if save_version is set to 105
prim		real	
ctrl	else if_cond		if save_version is greater than 105
prim		logical	"non_cross" or "cross"
prim		vector	magic
prim		logical	"non_smooth" or "smooth"
prim		logical	"non_smooth" or "smooth"
prim		real	Fullness

Description: This is all the VBL_SURF has to know about and provides a variety of different functions to do with computing the cross boundary derivative, setting up cached data, default relaxation methods etc.

BDY_GEOM_CIRCLE

Purpose: Implements the boundary geometry as an ellipse (actually a circle).

Derivation: BDY_GEOM_CIRCLE : BDY_GEOM : –

Data Elements:

sv id	curve type	Curve
prim	position	Center
enum	underlying_sf_type	Surface type enumeration
ctrl	if_cond	if the surface type is set to a torus
prim	position	Center
ctrl	if_cond	if the surface type is set to a pipe
sv id	surface type	Pipe surface
sv id	bs2_curve_def	Parameter space curve
prim	boolean	Reverse pcurve
prim	real	Pcurve fit tolerance
ctrl	if_cond	if the surface type is set to a given twist
prim	vector	Twist 0
prim	vector	Twist 1
prim	real	Start parameter
prim	real	End parameter
prim	boolean	Reverse

Description: This class implements the boundary geometry as an ellipse (actually a circle) which is a section of a cylindrical, toroidal or pipe surface, corresponding to the types CYLINDER, TORUS or PIPE. The only difference between these cases is how the twist vectors are calculated. This gives rise to the fourth case, GIVEN_TWIST, where it is no longer known what surface the circle really lies on but it is known what the relevant twist values are. As soon as the twists have been calculated any of the first three types could mutate themselves into a GIVEN_TWIST type.

BDY_GEOM_DEG

Purpose: This class implements a boundary section as a degenerate point.

Derivation: BDY_GEOM_DEG : BDY_GEOM : –

Data Elements:

prim	position	Position
prim	vector	Normal 0
prim	vector	Normal 1

Description: This class implements a boundary section as a degenerate point. For example, the “zero radius circle” which is imagined across an unblended edge. It will still represent an entire polygon edge in parameter space.

BDY_GEOM_PCURVE

Purpose: Defines all the functions the underlying geometry of the boundary curves must provide.

Derivation: BDY_GEOM_PCURVE : BDY_GEOM : –

Data Elements:

sv id	surface type	Surface on which pcurve lies
sv id	bs2_curve_def	Parameter space curve on surface
prim	boolean	Reverse pcurve
prim	real	Fit tolerance

Description: This implements the boundary geometry as a pcurve on the boundary surface.

BDY_GEOM_PLANE

Purpose: Implements the boundary geometry as a curve on a surface which must be a plane.

Derivation: BDY_GEOM_PLANE : BDY_GEOM : –

Data Elements:

prim	vector	Plane normal
prim	real	Start parameter for curve
prim	real	End parameter for curve
sv id	curve type	Curve on plane

Description: This implements the boundary geometry as a curve on a surface which must be a plane: we only need keep the unit normal (and a non-const pointer to the curve) in addition to the bounded curve. It is also used to represent a curve for which we have no normal data to interpolate.

“BEND”

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

Derivation: bend_law : multiple_law : law : –

Data Elements:

prim	string	The word “BEND” 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.

“bk”

Purpose: Links surface elements to curve elements.

Derivation: ATTRIB_SURFBACK : ATTRIB_MESH : ATTRIB : ENTITY : –

Data Elements:

sv id	“ms”	parent information
prim	$rec_num	Pointer to record in save file for most of the relevant link info ATTRIB_CURSURF
prim	$rec_num	Pointer to record in save file for remaining link info COMPCURV

Description: The back pointer attribute is used to link surface elements to curve elements via the curve element’s ATTRIB_CURSURF, which contains most of the relevant link information.

"bl_cr"

Purpose: Implements several trivial constant round geometries for edge-face and some face-face cases.

Derivation: ATT_BL_CR : ATT_BL_ENT_ENT : ATT_BL_ENT : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:			
	prim	real	Blend radius

Description: The geometry-making routines are provided for development only.

"bl_ent"

Purpose: A base class for a blend on some unspecified entities.

Derivation: ATT_BL_ENT : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:			
	prim	integer	Number of support entities
	ctrl	repeat	Repeat for the number of support entities
	prim	$rec_num	Pointer to record in save file for support ENTITY

Description: This base class represents a blend of some unspecified entities.

"bl_ent_ent"

Purpose: A base class for entity-entity blends.

Derivation: ATT_BL_ENT_ENT : ATT_BL_ENT : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:			
	enum	bl_convexity	text from blend convexity enumeration table
	prim	logical	either "unset" or "set"
	prim	position	help_pos

Description: Entity-entity blends process two spring curves in parallel to produce a chain of segments. All entity-entity blend classes are derived from this class.

"bl_inst"

Purpose: Defines the instruction attributes that hang onto the edges and vertices of a blend sequence, that instruct the blend algorithm.

Derivation: ATT_BL_INST : ATTRIB_BLINFO : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:

enum	transition_ents	text from blend transition enumeration table
prim	logical	either "unset" or "set" for position
prim	position	vertex position

Description: This class defines the attributes that provide special processing instructions to the blend algorithm.

"bl_seg"

Purpose: Defines the instruction attributes that hang onto the coedges and vertices of a sheet face to make the correct efinit and faceint attributes.

Derivation: ATT_BL_SEG : ATTRIB_BLINFO : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:

prim	No data	This class does not save any data

Description: Defines the instruction attributes that hang onto the coedges and vertices of a sheet face to enable the blend state 2 processing. These attributes are attached to each spring edge and cross curve edge made in the blend sheet. For spring edges, the start and end blend_ints have the same support.

"bl_support"

Purpose: Derived class for the blend support point.

Derivation: ATTRIB_BLEND_SUPPORT : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:

prim	$rec_num	pointer to
prim	$rec_num	pointer to previous support pointer
prim	$rec_num	pointer to next support pointer
prim	integer	support index data

Description: The blend support class is a pure base class. Extends this blend support. This is for use by constant round blend surfaces. They pass the newly extended spine and whether this is the left or right support.

"bl_taned"

Purpose: Defines a variable radius entity-entity blend.

Derivation: ATT_BL_TAN_ED : ATT_BL_VR : ATT_BL_ENT_ENT : ATT_BL_ENT : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

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

Description: Attribute for defining a variable radius entity-entity blend which is tangent to one surface along an edge and either goes through the other surface or is tangent to it.

"bl_vr"

Purpose: Defines a variable radius entity-entity blend.

Derivation: ATT_BL_VR : ATT_BL_ENT_ENT : ATT_BL_ENT : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:			
	sv id	curve (2) class	refer to curve object
	sv id	var_radius	refer to var_radius object, left object
	prim	newline	
	prim	boolean	if same_radii
	ctrl	if_cond	if the radii are not the same
	sv id	var_radius	refer to var_radius, right object
	prim	newline	
	sv id	var_cross_section	refer to var_cross_section object
	prim	newline	

Description: Attribute for defining a variable radius entity-entity blend. Very similar to the constant radius version, but we have a defining curve and a radius function instead of a single radius value.

“bldcur”

Purpose: This is one long edge of a general blend surface – a spring curve.

Derivation: blend_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
ctrl	if_cond		if save_version_number is less than INTCURVE_VERSION
sv id		bs3_curve_def	cur_data for curve
prim		real	fit tolerance
prim		newline	
sv id		surface type	surface 1
prim		newline	
sv id		surface type	surface 2
prim		newline	
sv id		bs2_curve_def	pcurve 1
prim		newline	
sv id		bs2_curve_def	pcurve 2
ctrl	else if_cond		else if save_version_number is less than PARCUR_VERSION
sv id		bs3_curve_save	cur_data for curve
prim		real	fit tolerance data
prim		newline	
sv id		surface type	surface 2
prim		newline	
sv id		surface type	surface 1
prim		newline	
sv id		bs2_curve_def	pcurve 2
prim		newline	
sv id		bs2_curve_def	pcurve 1
ctrl	else		
sv id		int_cur	
ctrl		if_cond	if save_version_number is greater than or equal to PARCUR_VERSION
prim		logical	“surf2” or “surf1”
prim	subtype_end		Right curly braces, “}” or Tag 16

Description: Defines an interpolated curve subtype which is a parameter line on a blend spline surface, for a parameterization where the true parameter line is everywhere within fit tolerance of the spline approximation’s parameter line.

This is a copy of the par_int_cur class, just with certain functions propped up to watch out for the zero length first derivatives which can occur at the ends the curve.

All the par_int_cur members are private rather than protected, so we could not just derive the new class from par_int_cur. surf1_data (in the parent) is used for the blend surface, and surf2_data for the other surface containing this curve.

“blend”

Purpose: Defines the basic blend attribute from which derive specific blend attributes.

Derivation: ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:	ctrl	if_cond	if the save version is less than CONSISTENT_VERSION
	prim	integer	Blend convexity data
	ctrl	else	if the save version is greater than CONSISTENT_VERSION
	enum	bl_cvxty_ents	Blend convexity data written as a string.
	prim	$rec_num	Pointer to record in save file for left face.
	prim	$rec_num	Pointer to record in save file for right face.
	prim	real	setback from the start.
	prim	real	setback from the end.
	ctrl	if_cond	if the save version is greater than or equal to ANG_XCUR_VERSION
	prim	real	setback diff at start.
	prim	real	setback diff at end.
	prim	logical	setback from the end.
	prim	logical	setback from the end.
	ctrl	if_cond	if the save version is greater than or equal to CONSISTENT_VERSION
	enum	bl_how_ents	How the blend was performed written as a string.
	ctrl	else	else, test for advanced blending version.
	ctrl	if_cond	if the save version is greater than or equal to ADV_BL_VERSION
	prim	integer	Specifies how the blend was created.
	sv id	surface type	Definition of surface

Description: This is an abstract class that tries to predict some of the fields that derived classes will need; for example, it contains pointers for a left surface, a left curve and a left point although in practice only one of these will be needed in a particular derived class. The reason for doing this is that the base class can completely handle the administrative functions such as save and restore, making these trivial for the derived classes. 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.

blend_spl_sur

Purpose: This class is used as a base class for all blend surfaces.

Derivation: 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_support	Left support
sv id	blend_support	Right support
sv id	curve type	Definition curve
prim	real	Left offset
prim	real	Right offset
prim	newline	
prim	logical	"single_radius" or "two_radii"
ctrl	left_rad–>save()	
ctrl	if_cond	if there are two_radii
ctrl	right_rad–>save();	
ctrl	section	
prim	newline	
prim	interval	*u*-param range
prim	interval	support *u*-param range
prim	interval	*v*-param range
prim	integer	Closed in *u*
prim	integer	Closed in *v*
prim	newline	
prim	subtype_end	Right curly braces, "}" or Tag 16

Description: This class defines the basic blend attribute from which derive specific blend attributes.

blend_support

Purpose: Represents the geometry on which a spring curve of a blend lies.

Derivation: blend_support : –

Data Elements:	prim	ident	Type name
	sv id	surface type	Surface
	sv id	curve type	Curve
	sv id	bs2_curve_def	Parameter space curve
	prim	position	Point

Description: It will be either a surface, a curve or a point. In general, only one of these will be stored in the class, although there is room for all three to be stored. This is to allow the administrative functions such as Save and Restore to be implemented just once, in the base class.

“blendsupcos”

Purpose: Derived class for the curve–on–surface case.

Derivation: blend_support_curve_on_surface : blend_support : –

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

Description: Extends this blend support. This is for use by constant round blend surfaces. They pass the newly extended spine and whether this is the left or right support.

“blendsupcur”

Purpose: Derived class for the blend support curve.

Derivation: blend_support_curve : blend_support : –

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

Description: Extends this blend support. This is for use by constant round blend surfaces. They pass the newly extended spine and whether this is the left or right support.

“blendsuppnt”

Purpose: Derived class for the blend support point.

Derivation: blend_support_surface : blend_support : –

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

Description: The blend support class is a pure base class. Extends this blend support. This is for use by constant round blend surfaces. They pass the newly extended spine and whether this is the left or right support.

"blendsupsur"

Purpose: Derived class for the blend support surface.

Derivation: blend_support_surface : blend_support : –

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

Description: The blend support class is a pure base class. Extends this blend support. This is for use by constant round blend surfaces. They pass the newly extended spine and whether this is the left or right support.

"blendsupzro"

Purpose: Derived class for the blend support zero curve.

Derivation: blend_support_zero_curve : blend_support : –

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

Description: The blend support class is a pure base class. Extends this blend support. This is for use by constant round blend surfaces. They pass the newly extended spine and whether this is the left or right support.

"blinfo"

Purpose: Defines attributes used internally by the blending algorithm to record intermediate results.

Derivation: ATTRIB_BLINFO : ATTRIB_SYS : ATTRIB : ENTITY : –

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

Description: This class defines attributes used internally by the blending algorithm to record intermediate results. It derive from a common base attribute to simplify clean up.

“blndsprngcur”

Purpose: This is one long edge of a general blend surface – a spring curve.

Derivation: spring_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	logical	Curve is left edge of blend
prim	subtype_end	Right curly braces, “}” or Tag 16

Description: surf1_data (in the parent) is used for the blend surface, and surf2_data for the other surface containing this curve.

“body”

Purpose: Represents a wire, sheet, or solid body.

Derivation: BODY : ENTITY : –

Data Elements:

ctrl	if_cond		if the save_version_number less than the LUMP_VERSION
ctrl		if_cond	if the lump is NULL
prim		$NULL	Pointer to NULL
ctrl		else	if the lump is not NULL
prim		$rec_num	Pointer to record in save file for first lump’s shell in body
ctrl	else		if the save_version_number greater than or equal to the LUMP_VERSION
prim		$rec_num	Pointer to record in save file for first lump’s shell in body
prim	$rec_num		Pointer to record in save file for first wire in body
prim	$rec_num		Pointer to record in save file for body transform

Description: A BODY models a wire, sheet, or solid body (possibly several disjoint bodies treated as one). The body contains zero or more wires, zero or more lumps, and a transform.

A pure *wire body* contains only wires, but no lumps, shells, or faces. Wires can represent isolated points, open or closed profiles, and also general wireframe models that are unpurified; i.e., have no faces.

A *lump body* represents partially-surfaced wireframes, sheets and solids. In a partially-surfaced wireframe, edges will border on zero, one or two faces, whereas edges in a sheet will meet one or two (or more) faces. In a manifold solid, every edge is adjacent to two faces and every vertex one group of adjacent faces (a nonmanifold solid has edges adjacent to more than two faces or more than one set of edge-adjacent faces at a vertex).

In principle, a body can contain both wire and lump components. Wire and lump components should not be connected to one another (in other words, there should be only one path from the body to any edge – via a wire or lump, but not both).

A solid is represented by the boundaries of the regions of space that it encloses. Each solid is made up of one lump or a group of disjoint lumps.

Lumps and wires are described in a local coordinate system. The local coordinate system is related to the global coordinate system by a transform stored with the body.

bounded_curve

Purpose: Defines a bounded curve.

Derivation: bounded_curve :–

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

Description: This class defines bounded curves. A bounded curve is a curve with a start and end parameters that specify the bounds of the curve. This class makes it easy to extract data from wireframe geometry. This class supports most of the functions, such as evaluation, curve length, etc., that are provided in the curve class.

“box”

Purpose: For internal use only.

Derivation: ATTRIB_BOX : ATTRIB_SYS : ATTRIB : ENTITY : –

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

Description: Refer to Purpose.

“bulletin”

Purpose: Holds the changes to a single ACIS entity.

Derivation: BULLETIN : –

Data Elements:	prim	$rec_num	old entity model
	prim	$rec_num	new entity model

Description: Maintains information relating to creation, modification, or deletion of an entity.

“bulletin_board”

Purpose: Holds all bulletins resulting from an API call.

Derivation: BULLETIN_BOARD : –

Data Elements:	prim	$rec_num	owner pointer
	prim	integer	status number
	ctrl	repeat	Repeat for all bulletins in bulletin board
	prim	integer	1: there is a next bulletin
	prim	newline	
	sv id	“bulletin”	save the individual bulletin
	prim	integer	0: there is no next bulletin

Description: When an entity in the ACIS model is created, altered, or deleted, a bulletin recording the data structure change is added to a bulletin board.

bs2_curve_def

Purpose: Defines a 2D B-spline curve.

Derivation: bs2_curve_def : –

Data Elements:	prim	ident		Type (“nullbs” “nurbs” or “nubs”)
	ctrl	if_cond		if the type is set to “nullbs”
				No curve saved
	ctrl	if_cond		if the type is not set to “nullbs”
	prim		integer	Degree
	prim		ident	Closure (“open” “closed” or “periodic”)
	prim		integer	Number of unique knot values
	ctrl		repeat	Repeat for the number of unique knot values
	prim			real	Knot value
	prim			integer	Knot multiplicity
	ctrl		repeat	Repeat for the (sum(Knot multiplicity)+1–degree)
	prim			real	*x*-coordinate of control point
	prim			real	*y*-coordinate of control point
	prim			if_cond	if the type is set to “nullbs”
	prim			real	Weight

Description: Definition of a 2D B-spline curve, used to approximate to true curves in the parameter space of a surface. This provides insulation between the modeler and the underlying spline library.

bs3_curve_def

Purpose: Provides an interface between ACIS and the underlying spline library.

Derivation: bs3_curve_def : –

Data Elements:	prim	ident		Type (“nullbs”, “nurbs”, or “nubs”)
	ctrl	if_cond		if the type is set to “nullbs”
				No curve saved
	ctrl	if_cond		if the type is not set to “nullbs”
	prim		integer	Degree
	prim		ident	Closure (“open” “closed” or “periodic”)
	prim		integer	Number of unique knot values
	ctrl		repeat	Repeat for the number of unique knot values
	prim			real	Knot value
	prim			integer	Knot multiplicity
	ctrl		repeat	Repeat for (sum(Knot multiplicity)+1–degree)
	prim			real	*x*-coordinate of control point
	prim			real	*y*-coordinate of control point
	prim			real	*z*-coordinate of control point
	prim			if_cond	if the type is set to “nullbs”
	prim			real	Weight

Description: This simply provides a definition for a spline curve type for the major part of ACIS to use.

bs3_surface_def

Purpose: Provides an interface between ACIS and the underlying spline library.

Derivation: bs3_surf_def : –

Data Elements:	prim	ident	Type (“nullbs” “nurbs” or “nubs”)
	ctrl	if_cond	if the type is set to “nullbs”
			No surface saved
	ctrl	if_cond	if the type is not set to “nullbs”
	prim	integer	*u*-degree
	prim	integer	*v*-degree
	ctrl	if_cond	if the type is set to “nurbs”
	prim	ident	Rational (“u” “v” or “both”)
	prim	ident	*u* Closure (“open” “closed” “periodic” or “unknown”)
	prim	ident	*v* Closure (“open” “closed” “periodic” or “unknown”)
	prim	ident	*u* Singularities (“none” “low” “high” or “both”)
	prim	ident	*v* Singularities (“none” “low” “high” or “both”)
	prim	integer	Number unique *u*-knots
	prim	integer	Number unique *v*-knots
	ctrl	repeat	Repeat for the number unique *u*-knots
	prim	real	*u*-knot value
	prim	integer	Knot multiplicity
	ctrl	repeat	Repeat for the number unique *v*-knots
	prim	real	*v*-knot value
	prim	integer	Knot multiplicity
	ctrl	repeat	Repeat for the number (sum(*v*-Knot multiplicity)+1–*v* degree)
	ctrl	repeat	Repeat for the number (sum(*u*-Knot multiplicity)+ 1–*u* degree)
	prim	real	*x*-coordinate of control point
	prim	real	*y*-coordinate of control point
	prim	real	*z*-coordinate of control point
	prim	if_cond	if the type is set to “nurbs”
	prim	real	Weight

Description: This simply provides a definition for a spline surface type for ACIS to use.

“camera”

Purpose: Used to save camera data in an ACIS part file.

Derivation: ACIS_CAMERA : ENTITY : –

Data Elements:

prim	position	camera position
prim	position	target
prim	vector	up vector
prim	real	view width
prim	real	view height
prim	real	near clipping plane
prim	real	far clipping plane
prim	real	center offset *x*
prim	real	center offset *y*
prim	real	image scale
prim	boolean	perspective

Description: An ACIS_CAMERA ENTITY is used to save camera data in an ACIS part file.

“capping_record”

Purpose: For internal use only.

Derivation: ATT_CAP_RECORD : ATTRIB_BLINFO : ATTRIB_SYS : ATTRIB : ENTITY : –

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

Description: Refer to Purpose.

“cap_ext”

Purpose: For internal use only.

Derivation: ATT_CAP_EXT : ATTRIB_SYS : ATTRIB : ENTITY : –

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

Description: Refer to Purpose.

"cap_info"

Purpose: For internal use only.

Derivation: ATTRIB_CAP_INFO : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:

prim	No data	This class does not save any data

Description: Refer to Purpose.

"cell"

Purpose: Attaches the subportion of a lump.

Derivation: CELL : ENTITY : –

Data Elements:

prim	$rec_num	Pointer to record in save file for next cell
prim	$rec_num	Pointer to record in save file for lump containing cell
prim	$rec_num	Pointer to record in save file for supercell containing cell
prim	boolean	Validity data

Description: The CELL is an abstract class for CELL2D and CELL3D. It provides the data and member functions for the list pointers, the lump back pointer, the supercell pointer, and the bounding box. It should never be instanced.

The cell is a minimal connected subportion of a lump. Usually a lump consists of a single cell,which is bounded by an outer cshell and zero, one, or more inner cshells representing voids within the cell. However, when an operation that leaves internal faces in a lump returns a lump with fully enclosed internal regions, each region is represented by a cell.

"cell2d"

Purpose: Connects the faces of a sheet.

Derivation: CELL2D : CELL : ENTITY : –

Data Elements:	prim	$rec_num	Pointer to record in save file for first CFACE in cell

Description: The cell2d is a maximal connected sheet. The CELL2D is a set of faces that are DOUBLE-SIDED and BOTH-OUTSIDE. It is directly descended from CELL.

“cell3d”

Purpose: Connects subportion of a lump.

Derivation: CELL3D : CELL : ENTITY : –

Data Elements:	prim	$rec_num	Pointer to record in save file for first CSHELL in cell

Description: The CELL3D is a three-dimensional sub-portion of a lump bounded by SINGLE-SIDED or DOUBLE-SIDED BOTH-INSIDE faces. Usually a lump consists of a single cell that is bounded by an outer cshell and zero, one or more inner cshells representing voids within the cell. However, when an operation that leaves internal faces in a lump returns a lump with fully enclosed internal regions, each region is a three-dimensional cell.

“cell_ptr”

Purpose: Implements an attribute describing a cell in the Cellular Topology Component.

Derivation: ATTRIB_CELL : ATTRIB_CT : ATTRIB : ENTITY : –

Data Elements:	prim	$rec_num	Pointer to record in save file for cell list
	prim	$rec_num	Pointer to record in save file for supercell list
	prim	boolean	Automatic update

Description: The ATTRIB_CELL attribute hangs from a lump and contains a pointer to the list of cells for that lump. It can optionally contain a list of supercells (a cell box hierarchy). An auto_update flag indicates to the automatic cell re-computation mechanism whether or not to update the cell data of the attribute. Each lump may have at most one ATTRIB_CELL.

"cface"

Purpose: A reference to one *side* of a FACE.

Derivation: CFACE : ENTITY : –

Data Elements:

prim	$rec_num	Pointer to record in save file for next CFACE in CELL2D or CSHELL
prim	$rec_num	Pointer to record in save file for owning CELL2D of CSHELL
prim	$rec_num	Pointer to record in save file for FACE that defines CFACE
prim	integer	Orientation of CFACE to face normal; REVBIT portion of integer

Description: A CFACE is a reference to one *side* of a face. If a face is double-sided, it should be pointed to by two cfaces, if it is single-sided, one cface uses it in the forward direction.

"cface_ptr"

Purpose: Defines an attribute that records two CFACE entities that refer to a given face.

Derivation: ATTRIB_FACECFACE : ATTRIB_CT : ATTRIB : ENTITY : –

Data Elements:

prim	$rec_num	Pointer to record in save file for front face
prim	$rec_num	Pointer to record in save file for back face

Description: Defines an attribute that is attached to a face and records at most two cface entities that refer to its front and back sides.

"cface_col_att"

Purpose: Assigns a color to a volume.

Derivation: ATTRIB_VOL_COL : ATTRIB_CFACE_VOL : ATTRIB_CT : ATTRIB : ENTITY : –

Data Elements:	enum	color	Color enumeration

Description: The ATTRIB_VOL_COL attribute is derived from ATTRIB_CFACE_VOL. This attribute assigns a color to a volume, and resides on all CFACEs of a three-dimensional cell. It also participates in CFACE volume attribute propagation.

“cface_vol_att”

Purpose: Used for volume attribute propagation member functions.

Derivation: ATTRIB_CFACE_VOL : ATTRIB_CT : ATTRIB : ENTITY : –

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

Description: Serves as a base class for attributes which should be propagated with volume modifications

“chamfer”

Purpose: Defines a flat chamfer blend.

Derivation: ATTRIB_CHAMFER : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:	prim	real	Left range
	prim	real	Right range

Description: This class defines a flat chamfer blend. An edge chamfer is a ruled surface between two spring curves, one on each lateral surface. Each spring curve is the perpendicular projection of a spine curve on to the appropriate lateral surface. The spine curve is the intersection of the two lateral surfaces offset by constant distances, which may be different for the two surfaces.

“coedge”

Purpose: Relates EDGEs with adjacent EDGEs and owning ENTITYs.

Derivation: COEDGE : ENTITY : –

Data Elements:	prim	$rec_num	Pointer to record in save file for next coedge in loop or wire
	prim	$rec_num	Pointer to record in save file for previous coedge in loop or wire
	prim	$rec_num	Pointer to record in save file for next coedge on edge
	prim	$rec_num	Pointer to record in save file for edge on which coedge lies
	ctrl	if_cond	if the save_version_number is less than the COEDGE_SENSE_VERSION
	prim	integer	Direction of coedge with respect to the edge enumeration
	ctrl	if_cond	if the save_version_number is greater than or equal to the COEDGE_SENSE_VERSION
	prim	logical	either "forward" or "reversed"
	ctrl	if_cond	if the save_version_number is less than the WIREBOOL_VERSION
	prim	$rec_num	Pointer to record in save file for shell owned by a wire. If shell also contains faces, then old_style data structures had the coedge owned by the shell, and no wire.
	ctrl	else	if the save_version_number greater than or equal to the WIREBOOL_VERSION
	prim	$rec_num	Pointer to record in save file for loop or wire to which coedge belongs
	prim	$rec_num	Pointer to record in save file for parameter space curve or geometry

Description: A COEDGE stores the relationships of the EDGE with adjacent EDGEs and superior owning ENTITYs. The data structures formed by the COEDGE pointers and their interpretation depends upon the nature of the owning ENTITY.

When the EDGE is part of a well-formed (manifold) solid body SHELL, it is adjacent to exactly two FACES. There are two COEDGEs, each associated with a LOOP in one FACE. In principle, the two FACEs could be the same, and even the LOOPs could be the same. All COEDGEs in a LOOP are linked into a doubly-linked list through next COEDGE and previous COEDGE pointers. The two COEDGEs for each EDGE are linked through their partner pointers (next coedge on edge). Several extensions to this arrangement are possible:

- A LOOP that is not closed has either a partially-defined or infinite FACE boundary. In this case, the next and previous lists are not circular, but terminate with NULL pointers.
- A SHELL that is not closed has free EDGEs at its boundary. For such EDGEs, there is only one COEDGE, with a NULL partner pointer.
- Nonmanifold SHELLs in which more than two FACEs meet in an EDGE result in the partner pointers for the COEDGEs (one for each FACE) being linked in a circular list.
- A WIRE body can be an object that may be a directed or undirected graph, made up of one or more disjoint WIREs, each of which is a collection of connected EDGEs. In this case, each EDGE has only one COEDGE, and COEDGEs are linked in circular lists around each VERTEX, using next and previous pointers according to which end of the COEDGE lies at the VERTEX.
- A SHELL can be of mixed dimensionality, containing both FACEs and unembedded EDGEs. The unembedded EDGEs are connected together as in WIREs, and are connected to any FACEs sharing their VERTEXes in a similar way. An additional rule is that in each FACE, the next list always yields the LOOP of EDGEs embedded in the FACE – any unembedded EDGE is connected via the previous pointer of one of the face COEDGEs. The unembedded EDGE can be reached from the FACEs through the VERTEX’s EDGE list of which the unembedded EDGE is a member.

“coinvert”

Purpose: Creates a coincident vertex attribute for stitching.

Derivation: ATTRIB_COINVERT : ATTRIB_SG : ATTRIB : ENTITY : –

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

Description: For internal use only.

"colour"

Purpose: User attribute class definition for a color.

Derivation: ATTRIB_COL : ATTRIB_TSL : ATTRIB : ENTITY : –

Data Elements:

enum	color		Color enumeration for entity display

Description: Implementation of color attribute. This is attached to body, face or edge objects, and is used to demonstrate attribute migration, as well as to produce prettier pictures in the ACIS test harness.

"comment"

Purpose: Constructs a PTC attribute.

Derivation: comment_attr_c : ATTRIB_SG : ATTRIB : ENTITY : –

Data Elements:

prim	integer			Data value
ctrl	if_cond			if the save_version_number is less than the CONSISTENT_VERSION
prim		integer		Number of characters
ctrl		repeat		Repeat for the number of characters
prim			integer	One character of comment (one char per value)
ctrl	else			if the save_version_number is greater than or equal to the CONSISTENT_VERSION
prim		ident		Whole comment string

Description: Include file for ACIS version of Pro Engineer text attributes.

com_cur

Purpose: Represents of a patchwork curve used to represent boundaries of a meshsurf, which is a patchwork surface.

Derivation: com_cur :–

Data Elements:	prim	logical	“forward” or “reversed”
	prim	newline	
	prim	integer	count of elements
	prim	integer	count of attributes
	prim	logical	“open” or “closed”
	prim	newline	
	sv id	“p1”	start node for that element, P1NODE
	ctrl	repeat	repeat until finished with elements
	sv id	“1d”	current element for class ELEM1D
	prim	newline	
	sv id	“p1”	end node for that element, P1NODE
	prim	newline	
	ctrl	repeat	repeat until finished with attributes
	sv id	“ms”	current attribute for class
	sv id	“cs”	current attribute for class
	sv id	curve (2) class	curve element

Description: This class defines the header for compcurv, which is the general representation of a patchwork curve used to represent boundaries of a meshsurf, which is a patchwork surface. Separate edges and vertices could have been used to represent this data, but the compcurv class is implemented to save memory, increase execution speed, represent parameter lines, and store attributes that refer to corresponding nodes and elements on the meshsurfs that they lie in.

“compcurv”

Purpose: Records a composite curve as a (lowercase) compcurv.

Derivation: COMPCURV : CURVE : –

Data Elements:	sv id	compcurv (2) class	start node for that element, P1NODE

Description: The COMPCURV class records a composite curve as a (lowercase) compcurv. The compcurv holds the arrays of coordinate nodes and elements (segments) that define a piecework curve.

Like other surface and curve types, COMPCURV contains a lowercase compcurv class object. Aside from the usual surface and curve entity class members, it has member functions to provide access to the nodes and elements that make up the surface and a binary tree of curve or surface bounding boxes.

The COMPCURV class contains a few specialized functions to help the Boolean operations process new composite curve intersections on mesh surfaces. The constructor is also specialized to process the CURVE back pointers that surface elements can contain when, for example, a curve is split.

compcurv (2) class

Purpose: Records a composite curve as a (lowercase) compcurv.

Derivation: COMPCURV : CURVE : –

Data Elements:

ctrl	if_cond	if save_version_number is less than the MESH_VERSION, there is an error
prim	logical	“forward” or “reverse”
sv id	com_cur	save the com_cur
sv id	curve (2) class	save the curve data

Description: The COMPCURV class records a composite curve as a (lowercase) compcurv. The compcurv holds the arrays of coordinate nodes and elements (segments) that define a piecework curve.

Like other surface and curve types, COMPCURV contains a lowercase compcurv class object. Aside from the usual surface and curve entity class members, it has member functions to provide access to the nodes and elements that make up the surface and a binary tree of curve or surface bounding boxes.

The COMPCURV class contains a few specialized functions to help the Boolean operations process new composite curve intersections on mesh surfaces. The constructor is also specialized to process the CURVE back pointers that surface elements can contain when, for example, a curve is split.

“cone”

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		CONE (1) class	derived from CONE 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		cone (2) class	derived from cone 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.

CONE (1) class

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

Derivation: CONE : SURFACE : ENTITY : –

Data Elements:

sv id	cone (2) class	Cone data definition given in another section of this manual.

Description: A CONE is defined by the base ellipse and the sine and cosine of the major half-angle.

cone (2) class

Purpose: Defines the elliptical single cone.

Derivation: cone : 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	cone_type; integer for type of cone
prim		integer	ellipse_type; integer for type of ellipse
ctrl		else	if save_version_number is greater than the SURFACE_VERSION
prim		string	save identifier; "cone".
sv id	ellipse (2) class		Base ellipse
prim	real		Sine of cone angle
prim	real		Cosine of cone angle
ctrl	if_cond		if the save_version_number is greater than or equal to the CONE_SCALING_VERSION
prim		double	Scaling *u* of parameter lines.
ctrl	if_cond		if the save_version_number is greater than or equal to the SURFACE_VERSION
prim		logical	*u* parameter reversed, either "forward" or "reversed"
sv id	surface (2) class		Generic surface data given in another section of this manual
ctrl	if_cond		if used as a subtype reference
prim		subtype_end	Right curly braces, "}" or Tag 16

Description: The cone class defines an elliptical single cone. It is defined by a base ellipse and the sine and cosine of the major half-angle of the core. The normal of the base ellipse represents the axis of the cone.

As special cases, the cross-section may be circular, the *cone* may be a cylinder, or both.

The polarity (sign) of the trigonometric functions define the slant of the surface of the cone and the sense of the surface.

- If sine_angle has different polarity than cosine_angle, the cross-section decreases in the direction of the axis of the cone (ellipse surface normal).
- If sine_angle has the polarity as cosine_angle, the cross-section increases in the direction of the axis of the cone (ellipse surface normal).

– If cosine_angle is positive (+), the sense of the surface is away from the axis of the cone (surface is convex). If cosine_angle is negative (–), the sense of the surface is toward the axis of the cone (surface is concave).
– If sine_angle is identically zero (sine_angle == 0), the cone is a cylinder.
– If cosine_angle is identically zero (cosine_angle ==0), the cone is planar.

The surface stops at the apex, if any; i.e., this surface type does not represent a double cone.

The *u*-parameter scaling factor stores the factor that when multiplied by the *u*-parameter of a point gives the 3D distance of that point along the cone surface from the base ellipse. The *u*-parameter is always 0.0 on the cone base ellipse. This enables the parameterization to be preserved if the cone is offset.

The *u*-parameter direction is along the generators of the cone, with zero representing the intersection of the generator with the base ellipse, and parameter increasing in the direction of the cone axis; i.e., the normal of the base ellipse, if reverse_u is FALSE, and in the opposite direction if reverse_u is TRUE. The *v*-parameter direction is along a cross-sectional ellipse clockwise around the cone axis, parameterized as for the base ellipse.

To evaluate the position corresponding to a given *uv* pair, first evaluate the base ellipse at parameter *v*, and subtract the center point to give vector *V*. Let *s* and *c* be sine_angle and cosine_angle if cosine_angle is positive, or –sine_angle and –cosine_angle if not. Let *R* be the length of the major axis of the base ellipse, negated if reverse_u is TRUE. Then:

```
pos = base.center + (1 + s*u)* V + c*u*R*base.normal
```

This parameterization is left-handed for a convex cone (cosine_angle > 0) with reverse_u FALSE or for a concave cone with reverse_u TRUE, and right-handed otherwise.

When the cone is transformed, the sense of reverse_u is inverted if the transform includes a reflection. A negation requires no special action.

In summary, cones are:

– Not true parametric surfaces.
– Are closed in *v* but not in *u*.
– Periodic in *v* (–pi to pi with period 2pi) but not in *u*.
– Singular in *u* at the apex; nonsingular for all other *u* and *v* values.

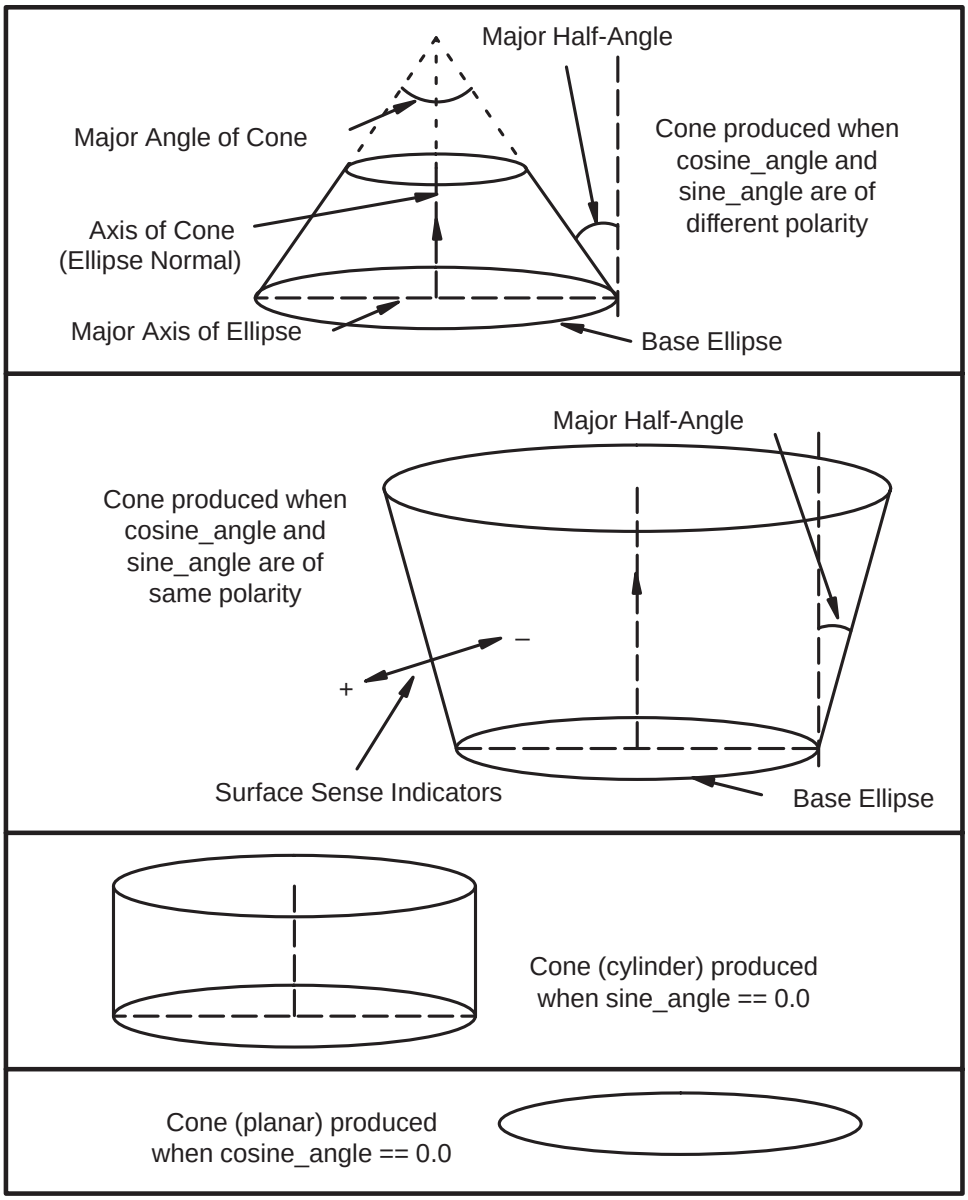


Figure 5-1. cone (2) class Definition

"const_chamfer"

Purpose: Defines a flat chamfer blend.

Derivation: ATTRIB_CONST_CHAMFER : ATTRIB_FFBLEND : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:	prim	real	Left range
	prim	real	Right range

Description: This class defines a flat chamfer blend. An edge chamfer is a ruled surface between two spring curves, one on each lateral surface. A spring curve is found by projecting the spine curve to each lateral face. The spine curve is the intersection of the two lateral surfaces offset by constant distances chosen to give the desired ranges which may be different for the two surfaces.

“const_round”

Purpose: Defines a circular rolling-ball blend of constant radius.

Derivation: ATTRIB_CONST_ROUND : ATTRIB_FFBLEND : ATTRIB_BLEND : ATTRIB_SYS : ATTRIB : ENTITY : –

Data Elements:	prim	real	Radius

Description: ATTRIB_CONST_ROUND is the blend attribute for constant radius. This attribute records a circular, rolling ball blend of constant radius.

constant

Purpose: Composes a law function that is a constant number.

Derivation: constant_law : law : –

Data Elements:	prim	string	A real number appears somewhere within this double quoted string.

Description: The character “#” or string “constant” should not appear in the creation of a law. Instead, the character “#” is meant to symbolize that any valid real number can be used as a constant. Whenever a number alone is given as a law as input to an operation, such as wire–body:offset, it does not need to be enclosed in quotation marks. It is assumed to be a law.

“convexity”

Purpose: For internal use only.

Derivation: ATTRIB_CONVEXITY : ATTRIB_SYS : ATTRIB : ENTITY : –

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

Description: Refer to Purpose.

"copy_marker"

Purpose: For internal use only.

Derivation: ATT_COPY_MARKER : ATTRIB_SYS : ATTRIB : ENTITY : –

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

Description: Refer to Purpose.

"COS"

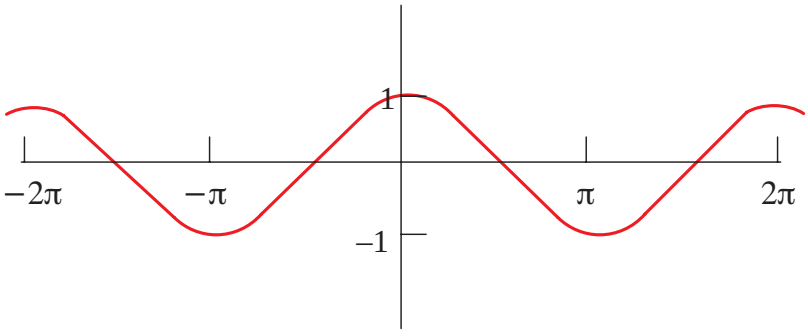
Purpose: Composes a law mathematic function that finds the cosine.

Derivation: cos_law : unary_law : law : –

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

Description: The mathematical definition is:

$y = \cos x$



"COSH"

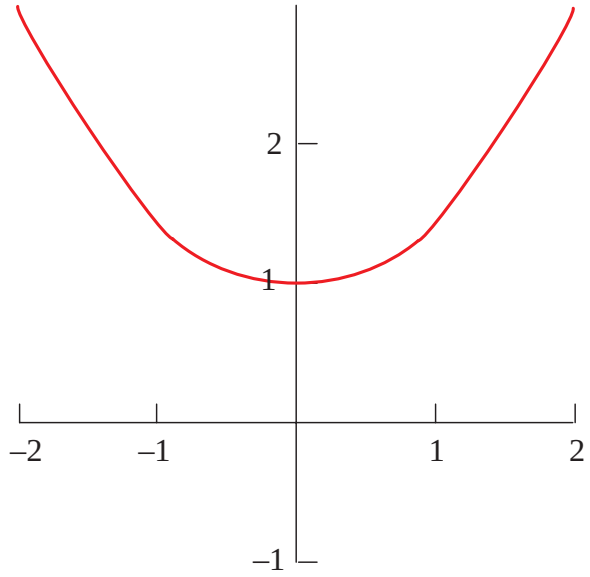
Purpose: Composes a law mathematic function that finds the hyperbolic cosine.

Derivation: cosh_law : unary_law : law : –

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

Description: The mathematical definition is:

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



"COT"

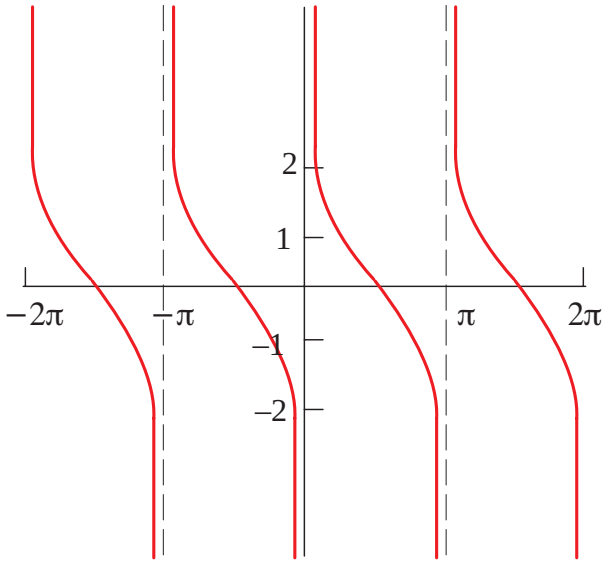
Purpose: Composes a law mathematic function that finds the cotangent.

Derivation: cot_law : unary_law : law : –

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

Description: The mathematical definition is:

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



“COTH”

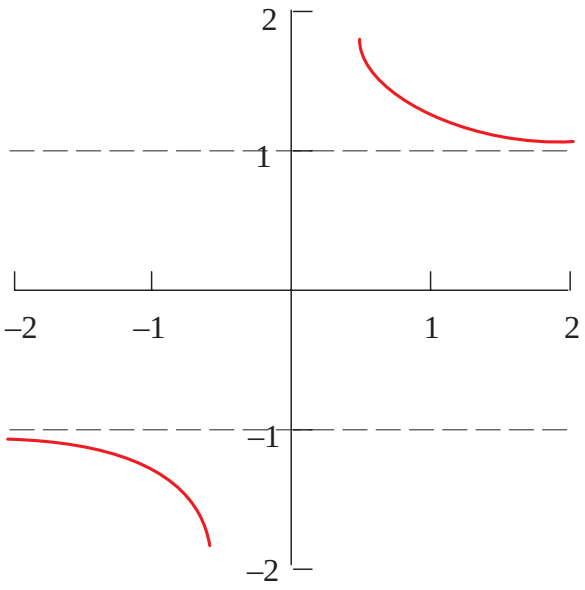
Purpose: Composes a law mathematic function that finds the hyperbolic cotangent.

Derivation: coth_law : unary_law : law : –

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

Description: The mathematical definition is:

$$y = \coth x = \frac{\cosh x}{\sinh x} = \frac{e^x + e^{-x}}{e^x - e^{-x}} \quad (x \neq 0)$$



"CROSS"

Purpose: Composes a law mathematic function that takes the cross product of two laws.

Derivation: cross_law : multiple_law : law : –

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

Description: This law symbol composes the law mathematic function that takes the cross product of two sublaws. The input sublaws, my_law1 and my_law2, must return 3 dimensional values. For example, these can be vec, cur, and wire. This only operates on laws; it only works on data types, such as gvector, if it is passed in as a law.

"crvcrvblndsur"

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

Derivation: crv_crv_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 the curves. The blend interpolates both curves, with no tangency constraints.

"crv_cstrn"

Purpose: Curve constraints.

Derivation: ATTRIB_CRV_CSTRN : ATTRIB_DSCSTRN : ATTRIB : ENTITY : –

Data Elements:			
	prim	real	numerical accuracy of tangent gain along the curve.
	ctrl	if_cond	if (acc_domain_crv_save)
	prim	long	acc_domain_crv_save–> Type_id( )
	sv id	curve type	acc_domain_crv_save–> Save_def( )
	ctrl	else	
	prim	long	–2: special flag

Description: For internal use only.

"crv_load"

Purpose: Curve loads.

Derivation: ATTRIB_CRV_LOAD : ATTRIB_DSLOAD : ATTRIB : ENTITY : –

Data Elements:	ctrl	if_cond	if (acl_domain_crv_save)
	prim	long	acl_domain_crv_save–> Type_id( )
	sv id	curve type	acl_domain_crv_save–>Save_def ( )
	ctrl	else	
	prim	long	–2; special case
	ctrl	if_cond	if (acl_src_pfunc_save)
	prim	long	acl_src_pfunc_save–> Type_id( )
	sv id	surface type	acl_src_pfunc_save–> Save_def( )
	ctrl	else	
	prim	long	–2 special case

Description: For internal use only.

“crvsrfblndsur”

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

Derivation: crv_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 a curve and a surface. The blend will be tangent to the surface, but have no tangency constraints where it interpolates the curve.

“cs”

Purpose: Implements the mesh curve and surface link attributes.

Derivation: ATTRIB_CURSURF : ATTRIB_MESH : ATTRIB : ENTITY : –

Data Elements:	sv id	"ms"	parent information
	prim	$rec_num	Pointer to record in save file for surface MESHSURF
	prim	$rec_num	Pointer to record in save file for partner attribute on surface ATTRIB_SURFBACK
	prim	$rec_num	Pointer to record in save file for starting node
	prim	$rec_num	Pointer to record in save file for ending node
	prim	integer	start side data
	prim	integer	end side data
	prim	integer	coincident side data

Description: The mesh curve and surface link attributes. These attributes interconnect mesh surface and curve elements, which are required to maintain their compatibility at all times. (Each surface node must map into a curve node and each curve element must fully span one element [or two if on their shared boundary]). The MESHSURF pointer in the attribute is used to obtain access to the higher level topology, which will be necessary during save, restore or copy.

"CSC"

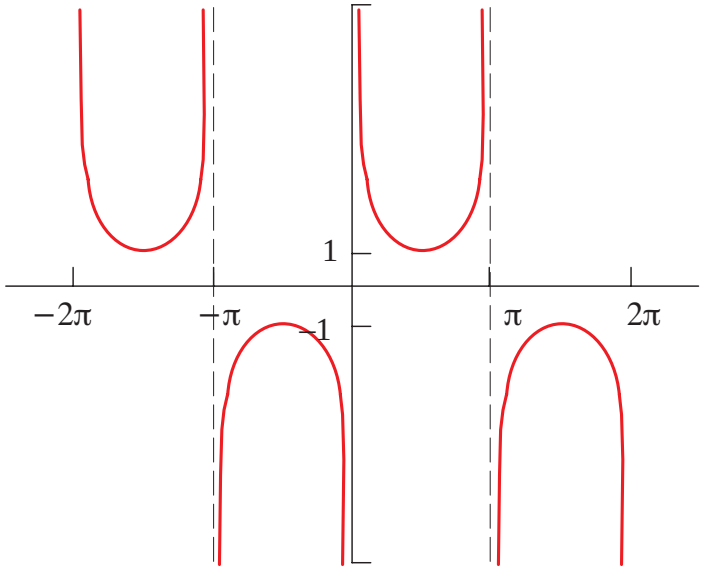
Purpose: Composes a law mathematic function that finds the cosecant.

Derivation: csc_law : unary_law : law : –

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

Description: The mathematical definition is:

$$y = \csc x = \frac{1}{\sin x}$$



"CSCH"

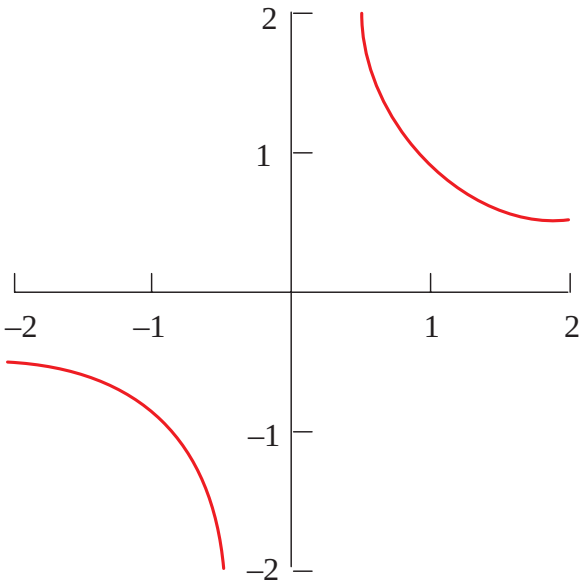
Purpose: Composes a law mathematic function that finds the hyperbolic cosecant.

Derivation: csch_law : unary_law : law : –

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

Description: The mathematical definition is:

$$y = \operatorname{csch} x = \frac{1}{\sinh x} = \frac{2}{e^{x} - e^{-x}} \quad (x \neq 0)$$



“cshell”

Purpose: One connected portion of a cell’s boundary.

Derivation: CSHELL : ENTITY : –

Data Elements:

prim	$rec_num	Pointer to record in save file for next CSHELL in CELL3D
prim	$rec_num	Pointer to record in save file for first CFACE in CSHELL
prim	$rec_num	Pointer to record in save file for owning CELL3D

Description: The CSHELL is one portion of a cell’s boundary; it is in a different shell from any other CSHELL in that cell. If a cell has no voids, then exactly one CSHELL gives its overall extent. Any other CSHELLs bound voids wholly within the cell. There is no distinction made in the data structure between peripheral and void CSHELLs, indeed, some CSHELLs may be open; e.g., a single face floating within a cell.

A CSHELL is constructed from a collection of CFACEs, each of which is one side of an existing face.

“ct”

Purpose: Defines a base attribute for the Cellular Topology Component.

Derivation: ATTRIB_CT : ATTRIB : ENTITY : –

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

Description: Defines a base attribute for the Cellular Topology Component. All Cellular Topology Component attributes are derived from this class.

“CUR”

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

Derivation: curve_law : unary_data_law : law : –

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

Description: cur returns the positions of the defining curve at the parameter value. In other words, this symbol is a way to pass in an edge into a law mathematic function for other purposes, such as evaluation. The dimension of the input, my_curve_law_data, is one, but when cur is evaluated, it returns an item in three dimensions.

ACIS defines its own parameter range for a curve which is not necessary the range [0,1]. If the mathematic function should be defined over the range [0,1], use the map law symbol.

“CURC”

Purpose: Composes a law mathematic function that returns the curvature of the curve at a parameter value.

Derivation: curvature_law : unary_data_law : law : –

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

Description: curc returns the curvature of the defining curve at the parameter value. This symbol operates only on sublaws, not on law data items. The dimension of the input, my_law, is one. When curc is evaluated, it returns a one dimensional item.

Unlike the cur and wire laws, the curc law symbol operations on laws and not on law data. Therefore, my_law must be a law that returns 3 values, as is the case for cur and wire law symbols.

ACIS defines its own parameter range for a curve which is not necessary the range [0,1]. If it should be defined over the range [0,1], use the map law symbol. The reciprocal of curvature is the radius of curvature.

"curve"

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		CURVE (1) class	derived from CURVE 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		curve (2) class	derived from curve 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.

CURVE (1) class

Purpose: Represents a variety of curve geometries as an object in the model.

Derivation: CURVE : ENTITY : –

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

Description: A CURVE provides the basic framework for the range of curve geometries implemented in the modeler. CURVEs may be used by more than one ENTITY.

curve (2) class

Purpose: Derives specific data used for the curve classes.

Derivation: curve : –

Data Elements: ctrl if_cond if the save_version_number is greater than or equal to the BNDCUR_VERSION

prim interval subset range

Description: The curve class is a base class from which all specific geometry classes (straight, ellipse, and intcurve) are derived. Considered as a function of its parameter, a curve is assumed to have a continuous first derivative whose length is bounded above and below by nonzero constants. There is no hard and fast rule about the values of these bounds or about the rate of change of the length of the derivative.

curve type

Purpose: Defines more specifically the type of curves.

Derivation: curve : –

Data Elements:	prim	ident	Curve type
	ctrl	if_cond	if Curve type is set to "null_curve"
			No curve saved
	ctrl	if_cond	if Curve type is set to "ellipse"
	sv id	ellipse (2) class	Ellipse definition
	ctrl	if_cond	if Curve type is set to "intcurve"
	sv id	intcurve (2) class	Intcurve definition
	ctrl	if_cond	if Curve type is set to "straight"
	sv id	straight (2) class	Straight definition
	ctrl	if_cond	if Curve type is set to "undefc"
	sv id	undefc (2) class	Undefc definition

Description: Refer to purpose.

"D"

Purpose: Composes a law mathematic function that takes one or more derivatives of a given law with respect to a given variable.

Derivation: derivative_law : multiple_law : law : –

Data Elements:	prim	string	The character "D" not part of another word and followed by something in parenthesis appears somewhere within this double quoted string.

Description: my_law specifies which law to take the derivative of. my_variable specifies what to take the derivative with respect to. n specifies how many derivatives to take. Only use this law symbol for derivatives that do not have exact derivatives. This returns the numerical derivative of the my_law.

"d5c2_cur"

Purpose: Defines an interpolated curve subtype which is a subtype of an int_cur.

Derivation: skin_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	integer	Number of parameters
	prim	newline	New line for readability in sat file
	ctrl	repeat	repeat for the number of parameters
	prim	real	specific parameter
	prim	newline	New line every fifth parameter
	ctrl	repeat	repeat for the number of parameters
	prim	real	specific parameter point *x*
	prim	real	specific parameter point *y*
	prim	real	specific parameter point *z*
	prim	newline	New line for readability in sat file
	ctrl	repeat	repeat for the number of parameters
	prim	real	specific parameter first derivative at *x*
	prim	real	specific parameter first derivative at *y*
	prim	real	specific parameter first derivative at *z*
	prim	newline	New line for readability in sat file
	ctrl	repeat	repeat for the number of parameters
	prim	real	specific parameter second derivative at *x*
	prim	real	specific parameter second derivative at *y*
	prim	real	specific parameter second derivative at *z*
	prim	subtype_end	Right curly braces, "}" or Tag 16

Description: Defines an interpolated curve subtype which is a subtype of an int_cur. This subtype is constrained to be a degree 5 spline with C2 continuity. It is defined in terms of positions, first and second derivatives at series of points. This curve is used exclusively as a temporary (working) geometry object, and should not appear in a BODY.

"dc_2acis"

Purpose: Connects the deformable surface functions to the ACIS modeler.

Derivation: ATTRIB_DC2ACIS : ATTRIB_DM2ACIS : ATTRIB_AGC : ATTRIB : ENTITY : –

Data Elements:			
	prim	No Data	This doesn't save any data

Description: The ATTRIB_DS2ACIS class encapsulates the programmable interface to deformable modeling. Using its methods, application programs can create, sculpt, and save deformable models. In addition, its methods support the API functions for managing the movement of data between ACIS FACE objects and deformable modeling objects.

This does not save any derived data – make sure those pointers are NULL because all the pointers in ATTRIB_DS2ACIS save all the graphics data

"DCUR"

Purpose: Composes a law mathematic function that finds the derivative of a given curve.

Derivation: dcurve_law : multiple_data_law : law : –

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

Description: my_curve_law_data is the curve used as input to the derivative operation. num specifies the number of derivatives to take of the curve. This is the same as cur except that it takes a second argument, num, for the number of derivatives. When num is 0, the result is the same as cur. This law symbol returns parametric derivatives as opposed to geometric derivatives.

"degenerate_curve"

Purpose: Used for skinning and lofting surfaces.

Derivation: degenerate_curve : curve : –

Data Elements:	ctrl	if_cond	if the save version number is less than CURVE_VERSION.
	prim	integer	degenerate curve type
	ctrl	else	
	prim	ident	writes a name string or NULL for the degenerate curve id.
	prim	position	root point
	sv id	curve type	save the curve information for the degenerate curve.

Description: The degenerate curve class was built for the skinning and lofting of surfaces. Its intended scope is to provide a way to build skin/loft surfaces that come to a point at either end.

“delete”

Purpose: For internal use only.

Derivation: ATTRIB_DEL : ATTRIB_SYS : ATTRIB : ENTITY : –

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

Description: Refer to Purpose.

“delta_state”

Purpose: Holds all bulletin boards resulting from modification to a model.

Derivation: DELTA_STATE : –

Data Elements:			
	prim	integer	Number for this state
	prim	integer	Number of states that can be rolled back
	prim	integer	hidden
	prim	$rec_num	previous delta state with respect to roll back
	prim	$rec_num	next delta state with respect to roll back
	prim	$rec_num	partner delta state; if no branches, points to itself.
	prim	$rec_num	delta state this is merged with
	prim	$rec_num	owner stream
	prim	ident	writes a name string or NULL
	ctrl	repeat	repeat for the number of bulletin boards
	prim	integer	1: a next bulletin board follows
	prim	newline	
	sv id	“bulletin_board”	save bulletin board information
	prim	integer	0: no more bulletin boards follow
	prim	newline	
	ctrl	if_cond	if there are merged_states
	prim	integer	number of iteration counts
	ctrl	repeat	Repeat for all merged states
	prim	$rec_num	merged state
	ctrl	else	
	prim	integer	0: no merged states
	prim	terminator	no more delta states

Description: A modification to a model may involve multiple API calls, which means multiple bulletin boards.

discontinuity_info

Purpose: Stores discontinuity information for a curve or surface.

Derivation: none

Data Elements:	prim	integer	Number of C1 discontinuities
	ctrl	repeat	Repeat for all C1 discontinuities
	prim	real	Parameter value of discontinuity
	prim	integer	Number of C2 discontinuities
	ctrl	repeat	Repeat for all C2 discontinuities
	prim	real	Parameter value of discontinuity
	prim	integer	Number of C3 discontinuities
	ctrl	repeat	Repeat for all C3 discontinuities
	prim	real	Parameter value of discontinuity

Description: Used to store parameter values at which a curve has a discontinuity in some derivative, or at which a surface has a line of discontinuity in some derivative.

“dict_entry”

Purpose: Stores names and their corresponding entities in an unordered linked list.

Derivation: DICT_ENTRY : ENTITY_TSL : ENTITY : –

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

Description: Simple name dictionary for the ACIS testbed. Simply stores names and their corresponding entities in an unordered linked list. Dictionary entries get logged and rolled back or forth with model. The name is placed in free store and a pointer to it recorded in the dictionary entry. Whenever the entry is copied for backup, the name is copied too, so is only used by one entity at a time.

“display_attribute”

Purpose: Defines DISPLAY_ATTRIB attribute to link ENTITYs with their display.

Derivation: DISPLAY_ATTRIB : ATTRIB_ST : ATTRIB : ENTITY : –

Data Elements:	prim	integer	display revision

Description: The class rendering_context_data is used in a DISPLAY_ATTRIB to link an ENTITY with its data in a specific rendering_context. An ENTITY can have more than one DISPLAY_ATTRIB attached to it. In fact, it will usually have one DISPLAY_ATTRIB for each rendering_context in which the ENTITY is displayed. The DISPLAY_ATTRIB serves to keep the display of an ENTITY up to date with its definition.

"dispose"

Purpose: Makes a face disposal attribute.

Derivation: ATTRIB_DISPOSE : ATTRIB_SYS : ATTRIB : ENTITY : –

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

Description: For internal use only.

"dist_press"

Purpose: Distributed pressure.

Derivation: ATTRIB_DIST_PRESS : ATTRIB_DSLOAD : ATTRIB : ENTITY : –

Data Elements:			
	prim	long	apr_domain_dim
	ctrl	repeat	Repeat for all apr_domain_dim
	prim	real	apr_domain_min[ii]
	prim	real	apr_domain_max[ii]

Description: For internal use only.

division

Purpose: Composes a law mathematic function that uses the division ("/") operator.

Derivation: division_law : binary_law : law : –

Data Elements:			
	prim	string	The character "/" appears somewhere within this double quoted string with elements preceding and following it.

Description: Parsing actually involves the "/" character. my_law1 and my_law2 can be any valid law mathematic function. my_law1 can have more than one dimension, but my_law2 has to be one dimensional.

“dm_2acis”

Purpose: Connects the deformable surface functions to the ACIS modeler.

Derivation: ATTRIB_DM2ACIS : ATTRIB_AGC : ATTRIB : ENTITY : –

Data Elements:

prim	No Data	This doesn’t save any data

Description: The ATTRIB_DS2ACIS class encapsulates the programmable interface to deformable modeling. Using its methods, application programs can create, sculpt, and save deformable models. In addition, its methods support the API functions for managing the movement of data between ACIS FACE objects and deformable modeling objects.

This does not save any derived data – make sure those pointers are NULL because all the pointers in ATTRIB_DS2ACIS save all the graphics data

“DOT”

Purpose: Composes a law mathematic function that takes the dot product of two vectors.

Derivation: dot_law : multiple_law : law : –

Data Elements:

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

Description: This law symbol composes the law mathematic function that takes the dot product of two sublaws. The input sublaws, my_law1 and my_law2, are not restricted in their return dimensional values. However, the input law with the smaller dimension is padded with zeros.

“ds_2acis”

Purpose: Connects the deformable surface functions to the ACIS modeler.

Derivation: ATTRIB_DS2ACIS : ATTRIB_AGC : ATTRIB : ENTITY : –

Data Elements:			
	prim	long	d2a_draw_state
	prim	real	d2a_surfC.red( )
	prim	real	d2a_surfC.green( )
	prim	real	d2a_surfC.blue( )
	prim	real	d2a_cptC.red( )
	prim	real	d2a_cptC.green( )
	prim	real	d2a_cptC.blue( )
	prim	real	d2a_cstrn_onC.red( )
	prim	real	d2a_cstrn_onC.green( )
	prim	real	d2a_cstrn_onC.blue( )
	prim	real	d2a_cstrn_offC.red( )
	prim	real	d2a_cstrn_offC.green( )
	prim	real	d2a_cstrn_offC.blue( )
	prim	real	d2a_loadC.red( )
	prim	real	d2a_loadC.green( )
	prim	real	d2a_loadC.blue( )
	prim	long	d2a_nu
	prim	long	d2a_nv
	prim	long	d2a_tag_count
	prim	real	d2a_comb_gain
	prim	long	d2a_comb_pt_count

Description: The ATTRIB_DS2ACIS class encapsulates the programmable interface to deformable modeling. Using its methods, application programs can create, sculpt, and save deformable models. In addition, its methods support the API functions for managing the movement of data between ACIS FACE objects and deformable modeling objects.

This does not save any derived data – make sure those pointers are NULL because all the pointers in ATTRIB_DS2ACIS save all the graphics data

"ds_cstrn"

Purpose: Deformable surface constraint.

Derivation: ATTRIB_DSCSTRN : ATTRIB_AGC : ATTRIB : ENTITY : –

Data Elements:			
	prim	long	acs_tag
	prim	long	acs_behavior

Description: For internal use only.

“ds_group”

Purpose: Connects the deformable surface functions to the ACIS modeler.

Derivation: ATTRIB_DSGROUP : ATTRIB_AGC : ATTRIB : ENTITY : –

Data Elements:

prim	$rec_num	Pointer to the base group record.
prim	$rec_num	Pointer to the Next group record.

Description: There exists one ATTRIB_DSMODEL attribute and one set of ATTRIB_DSGROUP attributes for every patch in a deformable model patch hierarchy. Each set of FACES or EDGES that map to a single deformable model DS_dmod object are associated to one another through the ATTRIB_DSGROUP attribute. Each such face is given one ATTRIB_DSGROUP attribute. At random, one of the ATTRIB_DSGROUP attributes in a group is made the ’base’ member. The ATTRIB_DSMODEL for the group is hung off of the base ATTRIB_DSGROUP object. The ATTRIB_DSGROUP object manages the logic of moving or losing the ATTRIB_DSMODEL object whenever it is lost. There may exist only one ATTRIB_DM2ACIS for an an entire patch hierarchy. The ATTRIB_DM2ACIS is attached to the base member of the root ATTRIB_DSGROUP set when it exists. The ATTRIB_DSGROUP callback contains the logic to decide whether to move or lose the ATTRIB_DM2ACIS model during merges of faces in the same ATTRIB_DSGROUP set. The root object is the one with no parent and at the end of the sibling list when more than one such attribute exists.

The deformable modeling group is implemented as a single linked list loop. Each member of the group contains a single pointer to the next member and a pointer to the base member. The base member differs from the rest of the members of the group because it has the ATTRIB_DSMODEL attribute. If the base member ceases to exist the ATTRIB_DSMODEL needs to be moved to the next member of the group and the base fields in each member need to be updated.

“ds_load”

Purpose: Deformable surface load.

Derivation: ATTRIB_DSLOAD : ATTRIB_AGC : ATTRIB : ENTITY : –

Data Elements:

prim	real	ald_gain
prim	long	ald_tag

Description: For internal use only.

"ds_model"

Purpose: Deformable surface model.

Derivation: ATTRIB_DSMODEL : ATTRIB_AGC : ATTRIB : ENTITY : –

Data Elements:

prim	long	adm_tag
prim	$rec_num	Pointer to the adm parent.
prim	$rec_num	Pointer to the adm sibling.
prim	$rec_num	Pointer to the adm child.
prim	long	adm_seam_state
prim	real	adm_au
prim	real	adm_av
prim	real	adm_atheta
prim	real	adm_bu
prim	real	adm_bv
prim	real	adm_btheta
prim	real	adm_delta
prim	real	adm_dt
prim	real	adm_mass
prim	real	adm_damp
prim	long	adm_ntgrl_degree
prim	long	adm_draw_state
prim	long	adm_mesh_u
prim	long	adm_mesh_v
prim	long	adm_comb_pt_count
prim	real	adm_comb_pt_gain

Description: For internal use only.

"DSURF"

Purpose: Composes a law mathematic function that finds the derivative of a given surface.

Derivation: dsurface_law : multiple_data_law : law : –

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

Description: my_surface_law_data is the curve used as input to the derivative operation. num specifies the number of derivatives to take of the curve. This is the same as surf except that it takes two more arguments. numu is the number of derivatives to take with respect to *u*, and numv is the number of derivatives to take with respect to *v*. When num is 0, the result is the same as surf. This law symbol returns parametric derivatives as opposed to geometric derivatives.

“DWIRE”

Purpose: Composes a law mathematic function that finds the derivative of a given wire.

Derivation: dwire_law : multiple_data_law : law : –

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

Description: my_wire_law_data is the wire used as input to the derivative operation. num specifies the number of derivatives to take of the curve. This is the same as wire except that it takes a second argument, num, for the number of derivatives. When num is 0, the result is the same as wire. This law symbol returns a scaled parametric derivatives as opposed to geometric derivatives.