Chapter 3. Functions

Topic: Ignore

The function interface is a set of Application Procedural Interface (API) and Direct Interface (DI) functions that an application can invoke to interact with ACIS. API functions, which combine modeler functionality with application support features such as argument error checking and roll back, are the main interface between applications and ACIS. The DI functions provide access to modeler functionality, but do not provide the additional application support features, and, unlike APIs, are not guaranteed to remain consistent from release to release. Refer to the *3D ACIS Online Help User's Guide* for a description of the fields in the reference template.

api_check_cur_smoothness

Function: Object Relationships

Action: Analyzes a curve for C1 or G1 discontinuities.

Prototype:

```
outcome api_check_cur_smoothness (
    EDGE* given_edge,           // edge to examine
    curve_irregularities*& cirr,// pointer to list of
                                // irregularities
                                // returned
    int& no_pts,                // number of
                                // irregularities
                                // returned
    AcisOptions* ao = NULL      // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/spline/sg_bs3c/bs3ccont.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Given an edge, this function examines the underlying curve and returns a list of parametric space points where there are C1/G1 discontinuities. The points are returned in a class called curve_irregularities.

The discontinuities can be of the following types:

– Tangents' magnitudes differ
– Tangents' directions differ

Errors: The pointer to an edge is NULL or does not point to an EDGE.

Limitations: This API works only for intcurves because other curves are always continuous.

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read-only

api_check_edge

Function: Debugging, Model Geometry, Model Topology

Action: Checks edge geometry for various conditions that could cause errors in other ACIS operations.

Prototype:

```
outcome api_check_edge (
    EDGE* edge,              // edge defined by a
                             // bs3_curve
    check_status_list*& list,// linked list of errors
                             // in edge returned
                             // (NULL if edge is OK)
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kernint/d3_chk/chk_stat.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API checks whether a edge defined by a bs3_curve is self–intersecting, is twisted, has too much oscillation, has degenerate edges, or is not G0, G1, or G2 continuous. The function list–>status can be checked for the results. If no errors are found, the NULL list is returned. For example:

```
while(list != NULL){
    switch(list->status()){
        case check_irregular:
            sys_error(CURVE_IRREGULAR);
            break;
        case check_self_intersects:
            sys_error(CURVE_SELF_INTER);
            break;
        case check_bad_closure:
            sys_error(CURVE_BAD_CLOSURE);
            break;
        case check_bs3_null:
            sys_error(BS3_CURVE_NULL);
            break;
        case check_bs3_coi_verts:
            sys_error(BS3_COI_VERTS);
            break;
        case check_bad_degeneracies:
            // This case never returned for curves
            break;
        case check_untreatable_singularity:
            // This case never returned for curves
            break;
        case check_non_G0:
            sys_error(CURVE_NON_G0);
            break;
        case check_non_G1:
            sys_error(CURVE_NON_G1);
            break;
        case check_non_G2:
            sys_error(CURVE_NON_G2);
            break;
        case check_non_C1;
            sys_error(CURVE_NON_C1);
            break;
        case check_inconsistent;
            printf(”Data mismatch in the given
edge”);
            break;
        case check_unknown:
            ;
    }
    list = list->next();
}
```

Errors:	None
Limitations:	None
Library:	intersct
Filename:	intr/intersct/kernapi/api/intrapi.hxx
Effect:	Read-only

api_check_entity

Function: Debugging, Entity, Model Topology, Model Geometry

Action: Checks an entity's geometry, topology, and data structure for errors.

Prototype:

```
outcome api_check_entity (
    const ENTITY* given_entity, // entity to be
                                // checked
    insanity_list*& list,       // list of insane
                                // entities and
                                // their info
    AcisOptions* ao = NULL      // ACIS options
    );
```

```
outcome api_check_entity (
    const ENTITY* given_entity, // entity to be
                                // checked
    ENTITY_LIST* problem_ents,  // list of insane
                                // entities and
                                // their info
    FILE* file_ptr = stdout,    //
    AcisOptions* ao = NULL      // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”intersct/sg_husk/sanity/insanity_list.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: A given entity and a pointer to insanity_list (refer to insanity_list class) are passed to this function. The given entity is traversed and checked for problems (see checks listed below). If any problem entities are found, the problem entities and the information are added to the insanity_list that is returned to the caller. NULL insanity_list indicates that the entity checks good.

A given entity, empty problem entity list, and, optionally, a file pointer are passed to the other function. The given entity is traversed and checked for problems (see checks listed below) and, if any problem entities are found they are added to the problem_ents list that is returned to the caller. The results of the check are written to the file defined by file_ptr, if provided, otherwise the results are written to the standard output device.

The option get_aux_msg can be used to control the level of detail for the information to be added to the list.

The messages corresponding to the insanity information for both the functions are tabularized in intersct/sg_husk/sanity/insanity_tbl.cxx and insanity_tbl.hxx. Users may change the insanity_tbl.cxx file to get customized insanity messages.

If the user wants to see the list of checks made when using either of the two functions, add the following lines of code to your program:

```
sanity_ctx& ctx = intersct_context()–>sanity();
printf(”\n”);
printf( ”checked:\n” );
printf( ”%d lumps\n”, ctx.lump_num );
printf( ”%d shells\n”, ctx.shell_num );
printf( ”%d wires\n”, ctx.wire_num );
printf( ”%d faces\n”, ctx.face_num );
printf( ”%d loops\n”, ctx.loop_num );
printf( ”%d coedges\n”, ctx.coedge_num );
printf( ”%d edges\n”, ctx.edge_num );
printf( ”%d vertices\n”, ctx.vertex_num );
if(ctx.law_num) printf( ”%d law\n”, ctx.law_num);
```

A sample output for the above code is as follows:

```
checked:
1 lumps
1 shells
0 wires
3 faces
3 loops
6 coedges
3 edges
2 vertices
```

An example usage of this API can be found in intr/intr_scm/chk_scm.cxx.

The level of checking and the output reported are controlled by the options check_level and check_output. If the check–level defined is not a multiple of ten, it is treated as the next lowest valid number. Any number over 70 is equal to 70. Entity checking is aborted after the first problem is found if the option check_abort is on.

The valid check levels are: 10 = Fast error checks
20 = Level 10 checks plus slower error checks (Default)
30 = Level 20 checks plus D–Cubed curve and surface checks
40 = Level 30 checks plus fast warning checks
50 = Level 40 checks plus slower warning checks
60 = Level 50 checks plus slow edge convexity change point checks
70 = Level 60 checks plus face–face intersection checks

The specific tests that api_check_entity performs include:

Level 20 Checks

Level 20 Checks			
Check Type	**S.No.**	**Check Description**	**Error Message**
BODY Checks	1.	Does the body have a non–NULL lump pointer?	body without lump
	2.	Does each lump in the body point back to the body?	body's lump does not point to body
TRANSFORM Checks	1.	Does the length of the translation vector exceed 2*SPAresabs/SPAresnor?	Transform translation is too large
	2.	If there are no rotation or reflection components in the transformation, is the affine portion of the transformation an identity matrix?	Matrix of non–rotation, non–reflection transform is not identity
	3.	Is each row vector and column vector of the affine portion of the transformation unit length?	Transform row or column is not a unit vector
	4.	If the transformation is a reflection, is the determinant of the affine portion of the transformation equal to –1.0?	Reflection transform determinant is not –1.0
	5.	If the transformation is not a reflection, is the determinant of the affine portion of the transformation equal to 1.0?	Transform determinant is not 1.0

Level 20 Checks			
ATTRIB Checks	1.	Given an entity with an attribute list, does the first attribute in the list have a non–NULL previous pointer (it should be NULL)?	attribute chain is corrupt
	2.	Given an entity with an attribute list, does each attribute in the list point back to the entity as the owner?	attribute owner is not correct
	3.	Given an entity with an attribute list, does each attribute in the list have the correct previous pointer; i.e. does it point to the attribute preceding it in the list?	attribute chain is corrupt
LUMP Checks	1.	Does the lump have a non–NULL shell pointer?	lump without shell
	2.	Does each shell in the lump have a non–NULL pointer to the lump?	lump's shell does not point to lump
	3.	Does the lump have a non–NULL pointer to its owning body?	lump without body
SHELL Checks	1.	Does the shell have a non–NULL face or wire pointer?	shell without face or wire
	2.	Do all of the faces in the shell have non–NULL pointers back to the shell?	shell's face does not point to shell
	3.	Do all of the wires in the shell have non–NULL pointers back to the shell?	shell's wire does not point to shell
	4.	Does the shell have a non–NULL pointer back to its owning lump?	shell without lump
WIRE Checks	1.	Does the wire have a coedge?	wire without coedge
	2.	Does each coedge in the wire have a non–NULL pointer back to the wire?	wire's coedge does not point to wire
	3.	For each coedge in the wire's list of coedges, is the end vertex of the current coedge contained in the next coedge?	coedge's next is not well connected

Level 20 Checks			
	4.	For each coedge in the wire's list of coedges, is the start vertex of the current coedge contained in the previous coedge?	coedge's previous is not well connected
	5.	Does the wire have a non–NULL owner pointer?	wire without owner
FACE Checks	1.	A face should normally have a non–NULL loop pointer pointing to a loop of coedges that bound the face. However, if the face is a complete sphere or torus it will have a NULL loop pointer (because there are no coedges) and will also be the only face in the shell. This check checks these different requirements and returns the error message below if any of them fail.	face without loop
	2.	A face's bounding box should be large enough so that it contains all of the face's vertices. For tolerant modeling, the face's bounding box should be large enough to contain all of the vertices after they have been expanded by the tolerance. This check checks to see if the bounding box is big enough for both of these cases and returns the error messages below upon failure. This problem can often be fixed by setting the face's bounding box to NULL so that it is recomputed upon the next request for the bounding box. This is done using the scheme command "entity:reset–boxes", the api function "api_reset_boxes(ENTITY *ent)", or "f–>set_bound(NULL)" where f is a pointer to a FACE object.	Face box too small, does not contain vertices For tolerant modeling:Face box with tolerant topology does not intersect vertex boxes

Level 20 Checks			
	3.	A full conical face (one with 0 radius at one end, i.e., has an apex) must have a loop at the apex that contains one coedge (and edge) with no geometric curve associated with it. There should be one vertex at the apex that is both the start vertex and end vertex for the edge. This check checks to make sure that these requirements are met for conical faces and returns the error message below if the check fails.	conical face needs NULL edge at apex
	4.	Does each loop in the face have a non–NULL pointer back to the face?	backptr to wrong face
	5.	Does the face have a non–NULL pointer to the shell it belongs to?	face without backptr
	6.	Does the face have a non–NULL pointer to the SURFACE model geometry?	face without SURFACE (ENTITY)
	7.	Does the SURFACE model geometry have a non–NULL pointer to the surface construction geometry?	face without surface (non–ENTITY)
	8..	Performs checks 1 – 13 in SURFACE check type.	
	9.	Performs checks 1 – 8 in PCURVES check type.	
	10.	Performs checks 14 – 16 in SURFACE check type.	
	11.	Performs checks 8 – 23 in LOOP check type.	
	12.	If the face's underlying surface is a plane, check that the face doesn't have zero or negative area	face with zero area face with negative area

Level 20 Checks			
	13.	If the face’s underlying surface is a cone, the loop checks indicate that it is not normally bounded (1 periphery loop or 2 u separator loops), and the only loops are not hole loops, check the face’s area to make sure it doesn’t have zero or negative area.	face with zero area face with negative area
	14.	Check that conical faces are bounded.	Warning Message: unbounded conical face
	15.	If the face’s surface is a non–degenerate torus, check the orientation by pushing points on the edge of the face toward the material side of the face and checking to see if they are contained in the face. If the loop is badly oriented then the points will get pushed in the wrong direction.	torus loop seems to be badly oriented
	16.	For multi–loop faces check the orientation of all of the loops and report the following error message when bad orientations are found.	face has loop with wrong orientation
SURFACE Checks	1.	If the surface is a plane, the normal is checked for unit length.	plane normal is not a unit vector
	2.	If the surface is a sphere, the radius is checked to insure it is not zero.	sphere has zero radius
	3.	If the surface is a cone, the cone’s base normal is checked for unit length.	cone’s ellipse normal is not a unit vector
	4.	If the surface is a cone, the base is defined using an ellipse. This check checks to make sure the ellipse’s major radius is not zero.	cone’s ellipse has zero length major axis

Level 20 Checks			
	5.	If the surface is a cone, the direction of the base normal is checked against the base ellipse major axis direction to insure that the cone axis (defined by the base normal) is perpendicular to the base.	cone's ellipse normal not perpendicular to major_axis
	6.	If the surface is a cone, the base is defined using an ellipse. The ratio of the base's minor axis radius to the major axis radius is checked to make sure it isn't 0.0.	cone's ellipse has zero radius ratio
	7.	If the surface is a cone, the base is defined using an ellipse. The ratio of the base's minor axis radius to the major axis radius is checked to make sure it isn't greater than 1.0.	cone's ellipse has radius ratio greater than 1.0
	8.	If the surface is a cone, the slope of the cone is specified using the sine and cosine of the major half–angle of the cone (see online ACIS documentation for "cone"). The sum of the squares of the sine and cosine is checked to insure it is 1.0.	cone's sine and cosine angle inconsistent
	9.	If the surface is a torus, the normal to the plane of the torus is checked to make sure it is a unit vector.	torus normal is not a unit vector
	10.	If the surface is a torus, the major radius is checked to make sure it is not zero.	torus major radius is zero
	11.	If the surface is a torus, the minor radius is checked to make sure it is not zero.	torus minor radius is zero

Level 20 Checks			
	12.	If the surface is a torus, the major radius is allowed to go smaller than the minor radius. As soon as it does, the torus overlaps itself in the center. As the major radius gets to zero the torus is a sphere. As the major radius goes negative, the torus will only have positive material until the major radius is smaller than negative the absolute value of the minor radius (absolute value since the minor radius can be negative). Therefore, this check checks to see if the major radius is greater than negative of the absolute value of the minor radius.	torus major radius less than negative of absolute value of minor radius
	13.	If the surface is a spline surface of type sub_spl_sur a warning is issued because these surface types may not behave correctly when attempts are made to extend them.	Warning Message: sub_spl_sur found – algorithms using surface extensions may fail
	14.	If the surface is a spline, is there a non–NULL bs3_surface?	Warning Message: spline surface has no approximating surface
	15.	If the surface is a spline, do the bs3_surface and the spl_sur have the same closure and periodic characteristics? Setting the ”check_fix” option to TRUE and rechecking can automatically fix this error.	procedural surface and its approximation have different forms

Level 20 Checks			
	16.	If the surface is a spline, this checks whether the closure and periodic characteristics in u and v of the bs3_surface are set correctly. It does this by sampling/comparing points on the surface. If it finds that the surface has incorrect closure or periodic characteristics it will give one of the following error messages. Setting the ”check_fix” option to TRUE and rechecking can automatically fix these errors.	bs3_surface form in u should be ’closed’ bs3_surface form in u should be ’open’ bs3_surface form in u should be ’periodic’ bs3_surface form in v should be ’closed’ bs3_surface form in v should be ’open’ bs3_surface form in v should be ’periodic’
	17.	If the surface is a spl_sur then this checks to see if there is an underlying bs3_surface.	no underlying bs3_surface
	18.	Checks to see if the bs3_surface is G1 continuous.	surface not G1
LOOP Checks	1.	Every loop should have at least one coedge so check for a non–NULL coedge pointer.	loop without coedge
	2.	Every coedge in a loop should point back to the loop as the owner.	loop’s coedge does not point to loop
	3.	The loop of coedges should be closed. Therefore, if there is only one coedge in the loop, its next pointer should point to itself. This is checked here.	loop back through next is not to start
	4.	The loop of coedges should be closed. Therefore, if there is only one coedge in the loop, its previous pointer should point to itself. This is checked here.	loop back through prev is not to start

Level 20 Checks			
	5.	The loop of coedges should be closed. Therefore, if you loop through all of the coedges in the list using the next pointer you should return back to the start coedge. Otherwise the loop is open and you get the error message below.	loop open in next() direction
	6.	The loop of coedges should be closed. Therefore, if you loop through all of the coedges in the list using the previous pointer you should return back to the start coedge. Otherwise the loop is open and you get the error message below.	loop open in previous() direction
	7.	Checks to make sure that the loop has a non–NULL pointer to its owning face.	loop without face
	8.	Checks if the face does not have any loops and the surface is not a sphere or torus.	error in face loop No periphery loop
	9.	Checks if there is a periphery loop and there is also a separator loop in u or v.	error in face loop Periphery exists with separation loop
	10.	Checks if there is a periphery loop that doesn't contain all the hole loops or all of the unknown loops.	error in face loop Hole is not contained
	11.	Checks if there are more than one periphery loops.	error in face loop More than one periphery loop
	12.	Checks to make sure that if there are no periphery loops and the surface is not a spline, then there must be no more than two separator loops.	error in face loop More than two separator loops
	13.	If the surface is a spline and there is only one loop, then it should be a periphery loop.	error in face loop No prop edge on periodic spline face
	14.	Checks if the surface is a torus and has only one v separator loop.	error in face loop Only one separation loop in torus

Level 20 Checks			
	15.	If the surface is a torus and has only one u separator loop, then it must be a degenerate torus with the major radius equal to the minor radius and the u separator loop must pass through the degenerate point at the center of the torus.	error in face loop Only one separation loop in torus
	16.	If the surface is a torus and has 2 u separator loops or 2 v separator loops, then the separator loops must mutually contain or exclude each other for a valid torus.	error in face loop Containment fails on separator
	17.	If the surface is a torus and has 2 u separator loops or 2 v separator loops and a non–zero number of hole loops, then the separator loops should both contain the hole loops.	error in face loop Hole is not contained
	18.	If the surface is a cone and it has one separator loop, then the sine_angle of the cone cannot be 0.	error in face loop Face is not completely bounded
	19.	If the surface is a cone and it has only one separator loop, then it must just be the apex of the cone.	error in face loop Face is not completely bounded
	20.	If the surface is a cone or a sphere and there are 2 u separator loops, then they must contain each other.	error in face loop Containment fails on separator
	21.	If the surface is a cone, then the separator loops must contain them.	error in face loop Containment fails on separator
	22.	Checks if no hole loops contain other hole loops.	error in face loop Hole contains hole
	23.	Check that for a spline surface no hole can exist without a periphery.	error in face loop No periphery loop
pcurve Checks	1.	Checks if the surface of the face that this pcurve resides on is a spline but the coedge does not have a pcurve.	coedge on spline surface has no PCURVE

Level 20 Checks			
	2.	Checks if the pcurve doesn’t have an underlying par_cur or bs2_curve.	pcurve has no defining curve
	3.	Checks if the pcurve is not periodic and its range does not include the range of its coedge.	Error Message for large discrepancy in ranges: pcurve’s range doesn’t include coedge’s range Warning Message for minor discrepancy in ranges: small difference in pcurve’s and coedge’s range can be neglected
	4.	Checks if point evaluation on the curve (associated with the edge) fails while checking the pcurve.	Warning Message: Could not complete pcurve checking due to curve problems
	5.	Checks if point evaluation on the pcurve using the ”eval” and ”eval_position” functions do not return the same parameter position on the surface.	There is a problem with the pcurve eval
	6.	If the pcurve’s surface is periodic but the face’s surface isn’t (after some operations there are some cases where the pcurve and face point to different surfaces), then this checks to see if the pcurve evaluates to (u,v) points in the correct period on the surface.	pcurve is a period outside of face range
	7.	Checks if the tangents along the pcurve go in roughly the same direction as the tangents along the curve. This is done by checking 4 points along the pcurve and curve.	pcurve direction != coedge direction

Level 20 Checks			
	8.	Calculates a position on the surface using the position on the pcurve as a guess for a point_perp() to the surface from the curve. Then it calculates the corresponding position on the curve and checks to see if the distance between the two positions is less than SPAresabs.	pcurve location != coedge location
COEDGE Checks	1.	Checks if the coedge has a non–NULL pointer to an edge.	coedge without edge
	2.	If a coedge represents an isolated vertex (the apex of a cone, for example), then the coedge should be the only coedge in the loop and its next and previous pointers should point to itself.	coedge for isolated vertex is embedded in loop
	3.	Checks if the coedge has a non–NULL pointer to its owner.	coedge without owner
	4.	Each edge has coedges associated with it. Checks if these coedges form a closed list of coedges that can be traversed by starting at the coedge pointed to by the edge and then traversing the list by getting the partner coedge of the current coedge until you get back to the coedge pointed to by the edge.	coedge's edge doesn't point to coedge
	5.	Checks if all the coedges in the closed list of coedges can be traversed by getting the partner of the current coedge point to the same edge.	coedge/partner edge mismatch
	6.	Check to make sure that a coedge's pointer to its partner does not point to itself.	coedge/partner points to itself

Level 20 Checks			
	7.	Check to make sure that the current coedge’s end vertex is the same as the next coedges’ start vertex and also that the current coedge’s start vertex is the same as the previous coedge’s end vertex.	sequential coedges do not share vertex
	8.	Check to make sure that a TCOEDGE is not associated with an EDGE.	TCOEDGE has a EDGE
	9.	Check to make sure that a COEDGE is not associated with a TEDGE.	COEDGE has a TEDGE
	10.	Check to make sure that a TCOEDGE has a pcurve associated with it.	TCOEDGE has no PCURVE
	11.	A TCOEDGE should have a PCURVE associated with it. The parameter range of the TCOEDGE should be contained within the parameter range of the PCURVE. This check makes sure that this is the case.	Pcurve range != tcoedge range
	12.	Checks to see if the start vertex of a coedge lies on the face to which it belongs.	start vertex not on face surface
	13.	Checks to see if the end vertex of a coedge lies on the face to which it belongs.	end vertex not on face surface
	14.	When tolerant modeling is done, this checks to see if the distance between the TCOEDGE’s start vertex position and the start position evaluated on the TCOEDGE using the start parameter (if available, the pcurve and surface are used; if not, the edge is used) is within the tolerance specified on the TCOEDGE’s start vertex.	Coedge start point outside tolerant vertex

Level 20 Checks			
	15.	When tolerant modeling is done, this checks to see if the distance between the TCOEDGE's end vertex position and the end position evaluated on the TCOEDGE using the end parameter (if available, the pcurve and surface are used; if not, the edge is used) is within the tolerance specified on the TCOEDGE's end vertex.	Coedge end point outside tolerant vertex
	16.	This check samples about 20 points along the edge pointed to by the coedge and insures that they are on the owning face.	internal position on coedge off face
EDGE Checks	1.	Checks if the edge has a non–NULL coedge pointer.	edge without backptr
	2.	Checks to see that the ordering of faces around an edge is correct (see ACIS online documentation on "Edges" under the section "Model Topology" for a description of the correct ordering). This is done by traversing the coedges on the edge and taking into consideration their sense as well as the sidedness and containment characteristics of the face associated with the current coedge. If the ordering is incorrect, then the error message is returned.	Coedges out of order about edge
	3.	Checks to see that the ordering of coedges around an edge is correct. This is done by making a copy of the current list of coedges (to save the original order), sorting the coedges based on the angle of each face to the first face, and then comparing the order of the newly sorted list to the original list. If the order doesn't match, then the error message below is returned.	Coedges out of order about edge

Level 20 Checks			
	4.	Check to ensure that the edge has a start vertex by checking for a non–NULL start vertex pointer.	edge without start vertex
	5.	Check to ensure that the edge has an end vertex by checking for a non–NULL end vertex pointer.	edge without end vertex
	6.	Check to make sure that if the edge is a TEDGE (tolerant modeling), then the parameter range of the edge is finite.	TEDGE has a NULL parameter range
	7.	In a manifold model, each vertex in the model will point to one edge and other edges attached to the vertex can be found by following the coedge network. Since all of the edges at the vertex in the manifold case belong to the same manifold group, this is sufficient. However, for non–manifold models where edges from different groups or manifolds may meet at a vertex, the vertex contains a pointer to an edge from each group. For example, if two cubes shared only a single vertex (non–manifold model), that vertex would contain a pointer to one edge from each cube or group (a total of two edge pointers). Checks to see if an edge is contained in more than one group of a single vertex.	vertex has edge in multiple groups
	8.	Each edge that has model geometry (CURVE) should also have construction geometry (curve). Checks for a non–NULL pointer.	edge without curve (non–ENTITY)

Level 20 Checks			
	9.	If the edge has a bounding box defined it should be large enough to include the start and end vertices. If it is not, then an error message is given. This problem can often be fixed by setting the face's bounding box to NULL so that it is recomputed upon the next request for the bounding box. This is done using the scheme command "entity:reset–boxes", the api function "api_reset_boxes(ENTITY *ent)", or the "EDGE::set_bound()" method.	Edge box too small, does not contain vertices
	10.	In tolerant modeling, if the edge has a bounding box it must intersect the start and end vertex boxes created by extending the vertex position by + and – the tolerance in all three directions. If it does not, then an error message is given. This problem can often be fixed by setting the face's bounding box to NULL so that it is recomputed upon the next request for the bounding box. This is done using the scheme command "entity:reset–boxes" or the api function "api_reset_boxes(ENTITY *ent)".	Tolerant Edge box does not intersect vertex boxes
	11.	Performs checks 1 – 8 in CURVES check type.	

Level 20 Checks			
	12.	Checks to see if the edge's start vertex lies on the curve by calculating the distance between the edge's start vertex and the point on the curve closest to the edge's start vertex. If the distance is greater than SPAresabs, then an error message is given.	start vertex not on curve
	13.	Checks to see if the edge's end vertex lies on the curve by calculating the distance between the edge's end vertex and the point on the curve closest to the edge's end vertex. If the distance is greater than SPAresabs, then an error message is given.	end vertex not on curve
	14.	Performs checks 9 – 14 in CURVES check type.	
	15.	Check to make sure that the curve's evaluation of the start point is the same as the edge's start vertex coordinates.	edge start param err
	16.	Check to make sure that the curve's evaluation of the end point is the same as the edge's end vertex coordinates.	edge end param err
	17.	Check to make sure that the edge's approximation curve has the same direction as the edge's true curve by evaluating the derivative of both at a midpoint on the curve.	approximating curve reversed
	18.	Performs checks 15 – 18 in CURVES check type.	
	19.	If the edge is a TEDGE (tolerant modeling) then this check checks to see if the curvature of the edge is tight enough anywhere that the edge with a tube, the radius of the tolerance, around it would intersect itself.	tolerant edge tube has local self–intersections

Level 20 Checks			
curve Checks	1.	If the curve is a straight line, then the direction vector of the curve is checked to make sure it is of unit length.	straight direction is not a unit vector
	2.	If the curve is a straight line, then the parameter scaling is checked to make sure it is not 0.0.	straight has zero value parameter scaling
	3.	If the curve is an ellipse, then the normal direction vector of the ellipse is checked to make sure it is a unit vector.	ellipse normal is not a unit vector
	4.	If the curve is an ellipse, then the major axis length is checked to make sure it is not 0.0.	ellipse has zero length major axis
	5.	If the curve is an ellipse, then the angle between the major axis and the normal is checked to make sure it is 90 degrees to insure the normal is perpendicular to the plane of the ellipse.	ellipse normal not perpendicular to major_axis
	6.	If the curve is an ellipse, then the ratio of the minor radius to the major radius is checked to make sure it is not 0.0.	ellipse has zero radius ratio
	7.	If the curve is an ellipse, then the ratio of the minor radius to the major radius is checked to make sure it is not greater than 1.0.	ellipse has radius ratio greater than 1.0
	8.	If the curve is defined by the intersection of two surfaces, then the pointers to the surfaces are checked to make sure they are non–NULL and that there is a valid construction geometry surface.	Intcurve has invalid supporting surface
	9.	If the curve is an intcurve, then check is made to make sure it has an int_cur associated with it.	spline curve has no approximating curve

Level 20 Checks			
	10.	If the curve is an intcurve, then the approximating curve is checked to make sure it has the same periodic and closedness characteristics as the procedural curve.	procedural curve and its approximation have different forms
	11.	If the curve is an intcurve, then the start and end points are checked to see if they are the same (closed curve). If they are the same but the curve is marked as ”open”, then an error message is given. Setting the ”check_fix” option to TRUE and rechecking can automatically fix this error.	closed bs3_curve marked as open
	12.	If the curve is an intcurve, then the start and end points are checked to see if they are the same (closed curve). If they are the same, then the derivatives at the ends are used to determine whether the curve should be considered periodic. If it is determined to be periodic but not marked, then an error message is given. Setting the ”check_fix” option to TRUE and rechecking can automatically fix this error.	periodic bs3_curve marked as closed
	13.	If the curve is an intcurve, then the start and end points are checked to see if they are the same (closed curve). If they are the same and the curve is not determined to be periodic but it is not marked as closed, then an error message is given. Setting the ”check_fix” option to TRUE and rechecking can automatically fix this error.	closed bs3_curve marked as periodic

Level 20 Checks			
	14.	If the curve is an intcurve, then the start and end points are checked to see if they are the same (closed curve). If they are not the same but the closed is not marked as ”open”, then an error message is given. Setting the ”check_fix” option to TRUE and rechecking can automatically fix this error.	open bs3_curve not marked as open
	15.	If the curve is an intcurve and has a non–NULL int_cur pointer, then the int_cur’s support surfaces are checked to make sure they have underlying bs3_surfaces.	A support surface with no underlying bs3_surface was found for the edge’s underlying intcurve
	16.	If the curve is an intcurve and has a non–NULL int_cur pointer, then the int_cur’s support surfaces are checked to make sure they are G1 continuous.	A support surface that is not G1 continuous was found for the edge’s underlying intcurve
	17.	If the curve is an intcurve and has a non–NULL int_cur pointer, then a check is made to make sure it has an approximating bs3_curve associated with it.	no underlying bs3_curve
	18.	If the curve is an intcurve and has a valid approximating bs3_curve, then the bs3_curve is checked for G1 continuity.	Warning Message: curve not G1
VERTEX Checks	1.	Checks to make sure that a vertex has a non–NULL pointer to geometry.	vertex without point
	2.	Checks to make sure that a vertex has a non–NULL pointer to an edge.	vertex without edge
	3.	Checks to makes sure that all edges pointed to by the vertex point back to the vertex.	vertex’s edge does not point to vertex

Level 20 Checks			
	4.	Checks makes sure that given a vertex and an edge attached to it, the coedges on the edge have the vertex as either a start or end point.	vertex/coedge closure mismatch. Fatal topology error. This part will hang.
	5.	While looping counter clockwise around a vertex examining coedges, if a coedge is pointing toward the vertex but the end vertex of the coedge does not equal the vertex used for looping, then an error message is given.	coedge CCW end vertex mismatch. Fatal topology error. This part will hang.
	6.	While looping counter clockwise around a vertex examining coedges, if a coedge is pointing away from the vertex and the coedge has a NULL previous pointer, then there is an invalid loop on the face that is not closed.	no CCW coedge closure around vertex. Fatal topology error. This part will hang.
	7.	While looping counter clockwise around a vertex examining coedges, if a coedge is pointing away from the vertex and the coedge has a previous pointer but the previous coedge's next coedge is not equal to the coedge pointing out, then there is invalid topology in the loop of coedges.	coedge CCW next/previous inconsistent. Fatal topology error. This part will hang.
	8.	While looping clockwise around a vertex examining coedges, if a coedge is pointing away from the vertex but the start vertex of the coedge does not equal the vertex used for looping, then an error message is given.	coedge CW start vertex mismatch. Fatal topology error. This part will hang.
	9.	While looping clockwise around a vertex examining coedges, if a coedge is pointing toward the vertex and the coedge has a NULL next pointer, then there is an invalid loop on the face that is not closed.	no CW coedge closure around vertex. Fatal topology error. This part will hang.

Level 20 Checks			
	10.	While looping clockwise around a vertex examining coedges, if a coedge is pointing toward the vertex and the coedge has a next pointer but the next coedge's previous coedge is not equal to the coedge pointing in, then there is an invalid topology in the loop of coedges.	coedge CW next/previous inconsistent. Fatal topology error. This part will hang.
	11.	This check loops CW and CCW around a vertex looking for problems with the coedge topology, in particular for cases where the chain of coedges around the vertex is not closed.	vertex chain open. Fatal topology error. This part will hang.
	12.	If there are any duplicate vertices, it is found by comparing the distance between two vertices to SPAresabs.	duplicate vertex
	13.	In tolerant modeling, overlapping vertices are checked for if at least one of the vertices is a TVERTEX. This is done by calculating the distance between the two vertices and comparing it to the largest of the two tolerances from the two vertices. If the distance is greater, the vertices are considered overlapping.	overlapping tolerant vertex
LAW Checks	1.	The definition of the law is checked to see if it is non–NULL.	internal law is NULL
	2.	Checks the approximation of the derivative of the law.	law derivative approximation error
Cellular Topology Checks	1.	Checks to make sure each cell points back to its lump.	Cell does not point to its lump
	2.	Checks to make sure each cell has a cshell.	Cell has no cshells
	3.	Checks to make sure each cell has been validated.	Cell is not validated
	4.	Checks to make sure that every cshell of a cell points to the cell.	Cshell does not point to its cell

Level 20 Checks			
	5.	Checks to make sure that every cshell has at least one cface.	Cshell has no cfaces
	6.	Checks to make sure that every cface points to its owner (usually a cshell).	Cface does not point to its owner
	7.	Checks to make sure that every cface points to a face.	Cface does not point to a face
	8.	Checks to make sure that every cface's face has an ATTRIB_FACECFACE attribute attached to it.	Cface points to face with no attrib
	9.	Checks to make sure that the ATTRIB_FACECFACE attribute attached to a cface's face points back to the cface in either the front_cface pointer or back_cface pointer.	Cface attrib does not point to cface
	10.	Checks to make sure that the cface is the front_cface in the ATTRIB_FACECFACE attribute but doesn't have an OUTSIDE sense.	Cface sense or attrib pointer is wrong
	11.	Checks to make sure that the cface is the back_cface in the ATTRIB_FACECFACE attribute but doesn't have an INSIDE sense.	Cface sense or attrib pointer is wrong
	12.	This check looks at each coedge on a cface and makes sure that an inward cface can be found that shares the coedge. If it cannot find one, then an error message is returned.	Cface has no closest inward on a coedge
	13.	If a double–sided outside face (which should be part of a CELL2D) is encountered while checking the cfaces of a CELL3D, then an error message is given.	Adjacent cfaces have inconsistent containments
	14.	If the closest inward cface to another cface is found but has a different owner, then the an error message is given.	Cface has closest inward with different owner

Level 20 Checks			
	15.	Checks to make sure that in a given lump every cshell is contained inside.	Cface has closest inward with different owner
	16.	If a ray fails to hit a cshell when checking that one cshell is contained in another, then an error message is given.	ray test failed to find intersection
	17.	Checks to make sure that all cshells in a lump contain each other.	Cshell does not contain other cshells
	18.	Checks to make sure that all 2D cells have at least one cface.	2D cell has no cfaces
	19.	All faces in the model are checked to make sure they have an ATTRIB_FACECFACE attribute attached to them.	Face has no ct attrib
	20.	Checks to make sure that all single–sided faces have a back cface.	Single sided face does not have back cface
	21.	Checks to make sure that all double–sided faces have a back cface.	Double sided face does not have back cface
	22.	Checks to make sure that if a double–sided face has a back cface, then the back cface has an owner.	Cface does not have owner
	23.	Checks to make sure that all double–sided faces have a front cface.	Double sided face does not have front cface
	24.	Checks to make sure that if a double–sided face has a front cface, then the front cface has an owner.	Cface does not have owner
	25.	Checks to make sure all of the cfaces are accounted for in the model hierarchy.	Some cfaces are not in hierarchy
	26.	If any errors are found while checking the lump cells, then an message is given.	Cellular Topology structure is corrupt. Is topology okay?

Note: Many of the errors with pcurves can be fixed by forcing the pcurves to be regenerated. The scheme command for doing this is:
entity:reset–pcurves
This scheme command executes the following functions:
sg_rm_pcurves_from_entity(ENTITY *ent, logical analytic_only=FALSE, logical do_tcoedges=FALSE)
(where analytic_only means only pcurves on analytic surfaces are to be removed and do_tcoedges means that pcurves from tolerant coedges are to be removed. The last parameter should never be set to TRUE)
and
sg_add_pcurves_to_entity(ENTITY *ent).

Data Structure Checks

– The parent has an appropriate child–level entity; e.g., body has lump
– Presence (non–NULL) and closure of back pointer from child to parent; e.g., body's lump points to body
– The coedge on a spline surface has a pcurve
– Pcurve indexing (0/+–1/+–2) is appropriate
– The pcurve has a non–NULL 2D B–spline curve
– If edge has a non–NULL curve, then the curve must have an equation

Topological Checks

– Loops are closed in both the next and previous directions
– Apex edge loops are correct
– Coedge has a partner, except an apex coedge
– All coedge partners point to the same edge
– Sequential coedges share a vertex
– Edge is in exactly one of start and end vertex edge groups. For example, edge can be reached for one value of i using start()–>edge(i)–>coedge() and partner and next (or previous) pointers.

Geometric Checks

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

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

- Spline surface form is set correctly, e.g. surfaces closed in *u* report this. Checks the underlying 3D B–spline surface at 10 points along seam to verify form.
- Checks that coedge vertices do not lie on spline surface singularities.
- Face normal is consistent with coedge direction, i.e.,
  a. Face area is greater than 0
  b. Multi–loop faces have loops correctly oriented
  c. All loops contain the others

- D3 surface checks (turned on by an option). Checks for a bad degeneracy – adjacent degenerate boundaries. The remaining checks are just on spline surfaces:
  a. surface irregular
  b. self intersection
  c. closure wrong
  d. no bs3_surface
  e. control point coincidence
  f. not C1, G0, G1, or G2

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

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

– Optional face–face intersection checking (option check_ff_int):
    a. Two valid faces have proper intersection:
        1. adjacent faces intersect only along a common edge
        2. nonadjacent faces do not intersect
    b. When a face–face Boolean fails, checks for proper edge–edge intersections on each face:
        1. adjacent edges intersection in common vertex only
        2. nonadjacent edges do not intersect
    c. Two valid, nonintersecting shells in the same lump have proper containment; i.e., each shell contains the other.
       A shell is valid if it contains no improperly intersecting faces.
    d. Two valid, nonintersecting lumps have proper containment, i.e., neither lump contains the other.
       A lump is valid when it contains no improperly intersecting faces and no shells with improper containment.

When face–face problems are found, a body containing the bad intersections is created. This body is named check_error.

Errors: The pointer to an entity is NULL.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read-only

api_check_face

Function: Debugging, Model Topology, Model Geometry

Action: Checks face geometry for various conditions that could cause errors in other ACIS operations.

Prototype:

```
outcome api_check_face (
    FACE* face,              // face defined by a
                             // bs3_surface
    check_status_list*& list,// linked list of errors
                             // in face returned
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kernint/d3_chk/chk_stat.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API checks whether a face defined by a bs3_surface is self–intersecting, is twisted, has too much oscillation, has degenerate edges, or is not G0, G1, or G2. The function list–>status can be checked for the results. If no errors are found, the NULL list is returned. For example:

```
while(list != NULL){
    switch(list->status()){
        case check_irregular:
            sys_error(FACE_IRREGULAR);
            break;
        case check_self_intersects:
            sys_error(FACE_SELF_INTER);
            break;
        case check_bad_closure:
            sys_error(SURF_BAD_CLOSURE);
            break;
        case check_bs3_null:
            sys_error(BS3_SURF_NULL);
            break;
        case check_bs3_coi_verts:
            sys_error(BS3_COI_VERTS);
            break;
        case check_bad_degeneracies:
            sys_error(BAD_DEGENERACIES);
            break;
        case check_untreatable_singularity:
            sys_error(UNTREAT_SING);
            break;
        case check_non_G0:
            sys_error(SURF_NON_G0);
            break;
        case check_non_G1:
            sys_error(SURF_NON_G1);
            break;
        case check_non_G2:
            sys_error(SURF_NON_G2);
            break;
        case check_non_C1:
            sys_error(SURF_NON_C1);
            break;
        case check_unknown:
            ;
    }
    list = list->next();
}
```

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read-only

api_check_wire_self_inters

Function: Object Relationships

Action: Evaluates a wire or wire body for self intersections.

Prototype:

```
outcome api_check_wire_self_inters (
    BODY* body,              // wire body to check
    AcisOptions* ao = NULL   // ACIS options
    );

outcome api_check_wire_self_inters (
    WIRE* wire,              // wire to check
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kerndata/top/wire.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Given a wire, this API examines the edges of the wire for intersections. Each edge of the wire is examined for self–intersections and for intersections with any of the other edges in the wire. As soon as any intersection other than shared end points is found, the API exits with an error outcome. If no such intersections are found, the API returns a successful outcome.

This function is overloaded. Given a wire body, this API examines the edges of the underlying wire for intersections.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read–only

api_create_boundary_field

Function: Intersectors

Action: Generates a vector field around a wire or face coedges.

Prototype:

```
outcome api_create_boundary_field (
    ENTITY_LIST& coeds,      // a list of edges
    FACE** ref_faces,        // reference faces for
                             // each edge, may be NULL
    fieldtype ftype,         // the type of vector
                             // fields to create
    int rev,                 // reverse flag for face
    double draft,            // draft angle from face,
                             // may be zero
    SPAvector& uniform_vec,  // reference vector to
                             // use (may be NULL_REF)
    ENTITY_LIST& cons_eds,   // constraint edges to
                             // setup coedge field
    int global,              // output vector fields
                             // in the global space
    law** out_law&,          // output laws for vector
                             // fields
    AcisOptions* ao = NULL   // ACIS options
    );

outcome api_create_boundary_field (
    ENTITY_LIST& coeds,      // a list of edges
    law** field_laws,        // reference laws to
                             // define basic fields,
                             // laws are in coedge
                             // local space
    double draft,            // draft angle from face,
                             // may be zero
    ENTITY_LIST& cons_eds,   // constraint edges to
                             // setup coedge field
    int global,              // output vector fields
                             // in the global space
    law** out_law&,          // output laws for vector
                             // fields
    AcisOptions* ao = NULL   // ACIS options
    );
```

```
outcome api_create_boundary_field (
    FACE* in_fas,           // create vector fields
                            // on coedges of this
                            // base face
    fieldtype ftype,        // type of vector fields
                            // to create
    int rev,                // reverse flag for face
    double draft,           // draft angle from face,
                            // may be zero
    SPAvector& uniform_vec, // reference vector to
                            // use (may be NULL_REF)
    ENTITY_LIST& cons_eds,  // constraint edges to
                            // setup coedge field
    int global,             // output vector fields
                            // in the global space
    law**& out_law,         // output laws for vector
                            // fields
    AcisOptions* ao = NULL  // ACIS options
    );

outcome api_create_boundary_field (
    WIRE* in_wire,          // the base wire on which
                            // to find coedges
    fieldtype ftype,        // the type of vector
                            // fields to create
    int rev,                // reverse flag for face
    double draft,           // draft angle from the
                            // face (may be 0)
    SPAvector& uniform_vec, // reference vector to
                            // use (may be NULL_REF)
    ENTITY_LIST& cons_eds,  // constraint edges to
                            // setup coedge field
    int global,             // output vector fields
                            // in the global space
    law**& out_law,         // output laws for vector
                            // fields
    AcisOptions* ao = NULL  // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/vector/vector.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”intersct/sg_husk/utils/coedfield.hxx”
#include ”lawutil/law_base.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kerndata/top/wire.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API generates vector fields on boundary coedges. These vector fields are represented in the form of ACIS laws and can be controlled by setting a target vector field type. The supported vector field types include:

– UNIFORM_VEC_FIELD
  Generate a constant vector fields for all coedges. Applications may supply a uniform vector. However, if no vector is supplied, an average wire normal vector or an average face normal vector will be used to define the uniform vector.
– FACE_NORMAL_FIELD
  Generated from the face normal that the coedge is on.
– FACE_TANGENT_FIELD
  Generated from the cross product of coedge tangent of face normal
– FACE_SIDE_FIELD
  If a base face has adjacent faces, the vector fields is generated from the cross product of the coedge tangent and the adjacent face normal.
– DRAFT_NORMAL_FIELD
  Generate vector fields that form a constant draft angle to the vector fields defined from either the UNIFORM_VEC_FIELD or the FACE_NORMAL_FIELD options.
– DRAFT_SIDE_FIELD
  Generate vector fields that form a constant draft angle to the vector fields generated from the FACE_SIDE_FIELD option.

The vector fields on G1 discontinuous coedges are adjusted so the fields will be continuous across vertices, i.e., no gaps between adjacent coedge vector fields. This capability allows applications to generate valid vector fields for subsequent operations such as lofting and sweeping.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read-only

api_crv_self_inters

Function: Object Relationships

Action: Locates self–intersections of a curve.

Prototype:
```
outcome api_crv_self_inters (
    CURVE* crv,               // curve which is
                              // interrogated
    double start_par,         // start parameter
    double end_par,           // end parameter
    curve_curve_int*&         // information
        self_intersection,    // corresponding to
                              // self-intersection
                              // returned
    AcisOptions* ao = NULL    // ACIS options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/geom/curve.hxx”
#include ”kernel/kernint/intcucu/intcucu.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API curve examines for points or intervals of self–intersection within the specified parameter range and results in the structure representing the intersection of two curves.

Use the following guidelines to interpret curve_curve_int information:

First, because the curve_curve_int record is meant for intersection of two curves, the information given for the second curve in the record is a duplication of the information given for first/original curve.

Second, if the curve_curve_rel is cur_cur_unknown, the intersection is an isolated point.

Third, if the curve_curve_rel is cur_cur_coin, the interval covered between this record and the next record (ignore the relation in the next record) gives the interval of overlap in the curve.

Except for the preceding relations, the cur_cur_int record for self–intersections does not contain any other relations. Also, it does not interpret the information in the record to obtain other relationships as it does for intersections of different curves.

Errors: None

Limitations: If the start parameter equals the end parameter, the API returns the point corresponding on the curve at given parameter in the self-intersection record.

If the curve given is NULL, this API does not return the isolated vertex point corresponding to an edge from which this NULL geometry might have come. To get that point, call api_ed_self_inters.

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_edent_rel

Function: Object Relationships

Action: Determines the relationship between a given edge and a given POINT, EDGE, FACE, or BODY.

Prototype:

```
outcome api_edent_rel (
    EDGE* edge,                 // the given edge
    ENTITY* entity,             // the given entity
    edge_entity_rel*& rel,      // the relationship
                                // between an edge and
                                // an entity
    AcisOptions* ao = NULL      // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”intersct/sg_husk/query/edentrel.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API determines the relationship between an edge and an entity.

The relationships typically indicate whether the edge lies inside, outside, or on the entity.

Errors: Pointer to edge is NULL or not to an EDGE.
Pointer to entity is NULL.

Limitations:	None
Library:	intersct
Filename:	intr/intersct/kernapi/api/intrapi.hxx
Effect:	Read–only

api_edfa_int

Function: Object Relationships, Intersectors

Action: Computes the intersections between the given edge and the given face.

Prototype:

```
outcome api_edfa_int (
    EDGE* edge,              // the given edge
    FACE* face,              // the given face
    ENTITY_LIST*& inter,     // the intersection
                             // returned in an entity
                             // list form
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API computes the intersections of a given edge with a given face. Then, it makes up the containment of the intersections and forms the edges or vertices, depending on whether the intersections are isolated or coincident. It returns the result in an entity_list as edges and vertices.

Errors: Pointer to edge is NULL or not to an EDGE.
Pointer to face is NULL or not to a FACE.

Limitations: This API may not give accurate results, or may even fail badly (i.e., segmentation violation), if the given face has an edge that is an inexact intcurve.

A work–around is to make SPAresabs equal to the fit tolerance with which the intcurve is fitted. However, the results are still not guaranteed.

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_edge_convexity_param

Function: Model Topology

Action: Analyzes the convexity of an edge at a given parameter value.

Prototype:

```
outcome api_edge_convexity_param (
    EDGE* edge,                // given edge
    double p,                  // parameter value at
                               // which to interrogate
    bl_ed_convexity& cxty,     // returned convexity
                               // value
    AcisOptions* ao = NULL     // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernbool/blending/bl_enum.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API takes an edge and a parameter value as the arguments and analyzes the convexity of the edge at that parameter value. The edge convexity reported as an enumerated type of type bl_ed_convexity.

A return value of either bl_ed_convex_smooth, bl_ed_concave_smooth, or bl_ed_smooth signifies a smooth edge.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read-only.

api_ed_self_inters

Function: Object Relationships

Action: Determines if a curve defines an edge for self–intersections.

Prototype:

```
outcome api_ed_self_inters (
    EDGE* edge,              // edge that is
                             // interrogated
    curve_curve_int*&        // information
        self_intersection,   // corresponding to
                             // self-intersection
                             // returned
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kernint/intcucu/intcucu.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API returns a curve_curve_int record containing the self–intersection information for the curve that defines the edge.

Use the following guidelines to interpret the curve_curve_int information.

Because the curve_curve_int record is meant for intersection of two curves, the information given for the second curve in the record duplicates of the information given for the first/original curve.

If the curve_curve_rel is cur_cur_unknown, the intersection is an isolated point.

If the curve_curve_rel is cur_cur_coin, the interval covered between this record and the next record (ignore the relation in the next record) gives the interval of overlap in the curve.

Except for above relations, the cur_cur_int record for self–intersections will not contain any other relations. Also, it will not interpret the information in the record to obtain other relationships as it will for intersections of two different curves.

Errors: NULL edge pointer.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read–only

api_entity_entity_distance

Function: Object Relationships

Action: Gets the minimum distance between two entities and the closest positions on those entities.

Prototype:

```
outcome api_entity_entity_distance (
    ENTITY* ent1,                // 1st entity
    ENTITY* ent2,                // 2nd entity
    SPAposition& pos1,           // 1st position
    SPAposition& pos2,           // 2nd position
    double& distance,            // distance value
    param_info& ent1_info        // parameter values
        =*(param_info*)NULL_REF, // on first entity
    param_info& ent2_info        // parameter values
        =*(param_info*)NULL_REF,// on second entity
    AcisOptions* ao = NULL       // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Using the two input entities, this API finds a position on each entity such that the distance between the two is the minimum distance. The distance as well as the two positions are returned. Supported entities include VERTEX, EDGE, LOOP, FACE, WIRE, SHELL, LUMP, and BODY.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read–only

api_entity_entity_touch

Function: Object Relationships

Action: Determines if two entities are “touching” (the distance between them is less than 2 * SPAresabs).

Prototype:

```
outcome api_entity_entity_touch (
    ENTITY* ent1,            // 1st entity
    ENTITY* ent2,            // 2nd entity
    logical& touch,          // touching or not
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Using the two input entities, this procedure determines if they are closer than 2 * SPAresabs, defaulting to find_entity_entity_distance if faster checks are inconclusive. If the distance is less than 2 * SPAresabs, it returns TRUE, otherwise FALSE. Supported entities include VERTEX, EDGE, LOOP, FACE, WIRE, SHELL, LUMP, and BODY.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read–only

api_entity_extrema

Function: Object Relationships

Action: Gets the extrema positions along the first given direction.

Prototype:

```
outcome api_entity_extrema (
    ENTITY_LIST& ents,       // list of entities on
                             // which to find extrema
    int nvec,                // number of directions
    SPAvector* in_vec,       // directions
    SPAposition& max_pos,    // extrema position
    param_info& out_info     // parameter info
        =*(param_info*)NULL_REF,
    AcisOptions* ao = NULL   // ACIS options
    );
```

```
outcome api_entity_extrema (
    ENTITY* ent,             // list of entities on
                             // which to find extrema
    int nvec,                // number of directions
    SPAvector* in_vec,       // directions
    SPAposition& on_pos,     // extrema position
    param_info& info         // parameter info
        =*(param_info*)NULL_REF,
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/vector.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: If there is more than one extrema position, the other directions will be used to determine a unique position. Optionally, the API will return the final entity and its parameter value. Applications may use this function to detect possible self–intersecting sweeping and to align lofting sections.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read–only

api_entity_point_distance

Function: Object Relationships

Action: Gets the minimum distance between an entity and a point and the closest position on the entity to the point.

Prototype:
```
outcome api_entity_point_distance (
    ENTITY* ent,                // entity
    SPAposition& in_point,      // point
    SPAposition& closest_pos,   // position on entity
    double& distance,           // distance value
    param_info& ent_info        // point on entity
        =*(param_info*)NULL_REF,// parameter info
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Using the input entity and point, this API finds a position on the entity such that the distance between the two is the minimum distance. The distance as well as the position is returned. Supported entities include VERTEX, EDGE, LOOP, FACE, WIRE, SHELL, LUMP, and BODY.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read–only

api_facet_curve

Function: Faceting

Action: Creates facets for a curve.

Prototype:
```
outcome api_facet_curve (
    const curve& crv,         // curve
    double a,                 // starting parameter
    double b,                 // ending parameter
    double tol,               // tolerance
    int nmax,                 // maximum number of
                              // points to generate
    int& npts,                // number of points
                              // generated (set to nmax
                              // +1 if nmax exceeded)
    SPAposition pts[],        // points on curve
                              // returned
    double t[],               // parameter value at
                              // point returned
    AcisOptions* ao = NULL    // ACIS options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerngeom/curve/curdef.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API retrieves a list of points and parameter values that approximate the given curve to within the specified tolerance.

Errors: Tolerance specified is not greater than zero.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read–only

api_face_nu_nv_isolines

Function: Silhouette and Isoparametric Curves

Action: Gets a list of *nu* (*nv*) *u* (*v*) edges that lie on the isoparametric curves and are trimmed to the boundaries of the face.

Prototype:

```
outcome api_face_nu_nv_isolines (
    int nu,                  // number of u-curves
    int nv,                  // number of v-curves
    FACE* face,              // face on which edges
                             // are calculated
    const SPAtransf& ftrans,// transformation
                             // positioning face
    ENTITY_LIST* edge_list, // list of edges returned
    AcisOptions* ao = NULL  // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/vector/transf.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API computes a *uv*-parameter box enclosing the face. The *u*-range is then subdivided into nu equal intervals and a *u*-isoparametric curve is calculated at the boundary of the intervals and trimmed to the boundaries of the face to get the edges. The *v*-iso curves are handled similarly. Because the parameter space box is not necessarily minimal, it is possible that the iso curve does not lie in the face and no edges are generated for this parameter value. The edges always have the associated transformation applied.

Errors: Pointer to face is NULL or not to a FACE.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_face_u_iso

Function: Silhouette and Isoparametric Curves

Action: Gets a list of edges that lie on a *u*-parametric curve and are trimmed to the boundary of a face.

Prototype:

```
outcome api_face_u_iso (
    double v,                    // v parameter
    FACE* face,                  // face on which edges
                                 // are calculated
    const SPAtransf& ftrans,// transformation
                                 // positioning face
    ENTITY_LIST* edge_list, // edges returned in list
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/vector/transf.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API computes the *u*-parametric curves on the surface of the face and trims them to the boundaries of the face to get the edges. It uses the face transformation to calculate the edges.

Errors: Pointer to face is NULL or not to a FACE.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_face_v_iso

Function: Silhouette and Isoparametric Curves

Action: Gets a list of edges that lie on a *v*-parametric curve and are trimmed to the boundary of a face.

Prototype:

```
outcome api_face_v_iso (
    double u,                    // u parameter
    FACE* face,                  // face on which edges
                                 // are calculated
    const SPAtransf& ftrans,// transformation
                                 // positioning face
    ENTITY_LIST* edge_list, // edges returned in list
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/vector/transf.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API computes the *v*-parametric curves on the surface of the face and trims them to the boundaries of the face to get the edges. It uses the face transformation to calculate the edges.

Errors: Pointer to face is NULL or not to a FACE.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_find_cls_ptto_face

Function: Object Relationships

Action: Determines the point on face nearest a given point in space.

Prototype:

```
outcome api_find_cls_ptto_face (
    const SPAposition& from_point,// point from which
                                 // the nearest point
                                 // is sought
    FACE* face,                  // face on which to
                                 // find the nearest
                                 // point
    SPAposition& to_point,       // nearest point
                                 // returned
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description:	Refer to Action.
Errors:	Pointer to face is NULL or not to a FACE.
Limitations:	None
Library:	intersct
Filename:	intr/intersct/kernapi/api/intrapi.hxx
Effect:	Read–only

api_get_ents

Function: Object Relationships, Ray Testing

Action: Gets a list of hits of a specified entity type by firing a ray at a body.

Prototype:

```
outcome api_get_ents (
    SPAposition const& ray_point,// start point of
ray
    SPAunit_vector const&       // direction of
        ray_direction,          // the ray
    double ray_radius,          // radius of ray
    int type_wanted,            // type of entities
                                // wanted
    BODY* target_body,          // body at which ray
                                // is to be fired
    ENTITY_LIST& entities_hit,  // entities hit by
                                // ray returned
    double*& ray_parameters,    // ray parameters
                                // returned
    AcisOptions* ao = NULL      // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/unitvec.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API fires a ray at a body from the given ray point in the given ray direction, with given ray radius and returns a list containing entities hit by the ray. Only the entities specified by type_wanted will be returned; all other hits are discarded.

The parameter value of the ray where each entity was hit is returned in ray_parameters. The caller is responsible for deleting this array.

A call to this API may cause boxes to be computed, changing the model and creating and a bulletin board. To make the process Read-only for the user, call api_note_state before the call to api_get_ents, then do

```
DELTA_STATE* ds = NULL;
api_note_state(ds);
api_change_state(ds);
api_delete_ds(ds);
```

to roll the modeler back to its state before api_get_ents was called.

Errors: Pointer to body is NULL or not to a BODY.
Zero length ray direction vector specified.
Ray radius not greater than zero.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_initialize_intersectors

Function: Modeler Control, Intersectors

Action: Initializes the intersector library.

Prototype:
```
outcome api_initialize_intersectors ();
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: System routine

api_intersect_curves

Function: Model Geometry, Construction Geometry

Action: Computes all intersections of two curves.

Prototype:
```
outcome api_intersect_curves (
    EDGE* crv1,              // first curve
    EDGE* crv2,              // second curve
    logical bounded,         // find only
                             // intersections within
                             // bounds
    curve_curve_int*& inters,// intersections
                             // returned
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kernint/intcucu/intcucu.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API returns a list of all intersections between the specified curves. If bounded is TRUE, the API returns only intersections within the parameter ranges of both curves (crv1 and crv2); otherwise, it returns intersections that occur on the extensions of the curves.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read-only

api_inter_ed_ed

Function: Intersectors

Action: Intersects two coplanar edges, producing a list of edge–edge intersection records.

Prototype:

```
outcome api_inter_ed_ed (
    EDGE* e1,                // first edge
    EDGE* e2,                // second edge
    curve_curve_int*& inters,// list of intersection
                             // information returned
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kernint/intcucu/intcucu.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: The records contain the locations of the intersection and the nature of the intersection (coincident, etc.).

To turn the list of intersection records into the POINTs and EDGEs of the intersection, use api_ed_inters_to_ents.

Where one edge is analytic, create the intersection by creating a corresponding analytic surface containing the curve and using the kernel surface–curve intersectors.

Spline–spline intersection goes directly to the spline package.

Errors: Pointer to edge is NULL or not to an EDGE.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_point_in_body

Function: Object Relationships

Action: Determines whether the given point lies inside, outside, or on the boundary of a given body.

Prototype:

```
outcome api_point_in_body (
    SPAposition const& test_point,// point to test
    BODY* target_body,            // body of interest
    point_containment& pc,        // inside, outside,
                                  // boundary, unknown
                                  // returned
    logical use_boxes = TRUE,     // Use bounding boxes
    AcisOptions* ao = NULL        // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”intersct/kernapi/api/ptcont.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
#include ”baseutil/logical.h”
```

Description: This API tests the point (given in global body space) against the body, returning a point_containment value of point_inside, point_outside, point_boundary, or point_unknown.

A call to this API will cause boxes to be computed, causing a model change and the generation of a bulletin board.

If the logical use_boxes is TRUE, bounding boxes will be considered increasing performance when the body is not a void. If there is a chance the body is a void, set to flag to FALSE so classification is correct.

To make the process Read-only, call api_note_state before the call to api_point_in_body. Afterwards, to roll the modeler back to its state prior to api_point_in_body perform:

```
DELTA_STATE*ds = NULL
api_note_state(ds);
api_change_state(ds);
api_delete_ds(ds);
```

Errors: Pointer to body is NULL or not to a BODY.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_point_in_face

Function: Object Relationships

Action: Determines the relationship of a position within a given face’s surface

Prototype:

```
outcome api_point_in_face (
    SPAposition const& test_point,// point in
                                  //question
    FACE* test_face,          // face to test point
                              // against
    SPAtransf const& face_trans,// transformation of
                                // the face
    point_face_containment& cont_answer,//
                              // relationship found
                              // (point_inside_face,
                              // point_outside_face,
                              // point_boundary_face,
                              // and
                              // point_unkown_face)
    SPApar_pos const& test_uv_guess,// spline face
                              //case,
                              // guess of
                              // uv-coordinates of the
                              // testpoint
    SPAposition const& prev_point_input,// previous
                                        // point
    point_face_containment prev_cont,// previous
                              // point’s containment
    logical use_cache,        // use cached entries
    int cache_size,           // size of array used to
                              // cache interior points
    AcisOptions* ao           // options such as
                              // journaling and
                              // versioning
    );
```

```
outcome api_point_in_face (
    SPAposition const& test_point,// point in
                                 //question
    FACE* test_face,             // face to test point
                                 // against
    SPAtransf const& face_trans,// transformation of
                                 // the face
    point_face_containment& cont_answer,//
                                 // relationship found
                                 // (point_inside_face,
                                 // point_outside_face,
                                 // point_boundary_face,
                                 // and
                                 // point_unkown_face)
    SPApar_pos const& test_uv_guess = // Spline face
        *(SAND(SPApar_pos) *)NULL_REF // case,
                                 // uv-coordinates guess
                                 // of the testpoint
    logical use_cache,           // use cached entries
    int cache_size,              // size of array used to
                                 // cache interior points
    AcisOptions* ao              // options such as
                                 // journaling and
                                 // versioning
    );
```

Includes:

```
#include ”baseutil/debug/module.hxx”
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/check.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”intersct/sg_husk/api/intr_jour.hxx”
#include ”intersct/kerndata/ptinface/ptinface.hxx”
```

Description: This function returns the containment of a given point for a given face. There are four possible outcomes: point_inside_face, point_outside_face, point_boundary_face, and point_unkown_face.

The transformation will be applied only if specified. Typically this is the transformation from the body that contains the face.

The test_uv_guess argument provides additional help by providing a coordinate in parametric space for an initial guess on where the position may lie. This may help to improve the performance.

If a previous position is specified, a containment description must be provided (point_inside_face, point_outside_face, point_boundary_face or point_unkown_face). This position can help find the position containment on the face at a faster rate. It is recomended to use a position that was previously found using this function. If the previous position passed in is a NULL REF it will call the other api_point_in_face function.

If api_point_in_face is called often on the same face, then setting use_cache to TRUE will increase the performance. If api_point_in_face is called with the use_cache set to TRUE an attribute will be attached to the face storing the cached information. If api_point_in_face is called subsequently on a face with the cached attribute information, but the use_cache flag is set to FALSE, the information will be ignored.

Cache size refers to how many levels of cache are used to store and search previously found data. Default is 10.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/ptinfc.cxx

Effect: Read–only

api_ptent_rel

Function: Object Relationships

Action: Determines the relationship of given point to a given entity (POINT, EDGE, FACE, or BODY).

Prototype:

```
outcome api_ptent_rel (
    APOINT* point,           // given point
    ENTITY* entity,          // given entity
    point_entity_rel*& rel,  // relationship between a
                             // point and an entity
                             // returned
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”intersct/sg_husk/query/ptentrel.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kerndata/geom/point.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API determines the relationship between a point and an entity. Typically, the relationship indicates whether the given point lies on, inside, or outside of the given entity.

Errors: Pointer to point is NULL or not to an APOINT.
Pointer to entity is NULL.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Read only

api_raytest_body

Function: Object Relationships, Ray Testing

Action: Gets the list of hits when a ray is fired at a body.

Prototype:

```
outcome api_raytest_body (
    SPAposition const& ray_point,// starting point of
                                // the ray
    SPAunit_vector const&       // direction of
        ray_direction,          // the ray
    double ray_radius,          // radius of the ray
    int hits_wanted,            // number of hits
                                // requested
    BODY* target_body,          // target body
    hit*& hit_list,             // list of entities
                                // hits along the ray
                                // returned
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”intersct/kerndata/raytest/raytest.hxx”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/unitvec.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API fires a ray at a body from the given ray point in the given ray direction with given ray radius. It returns a list that contains the *n* hits nearest the ray point (where *n* is the number of hits recorded). Only entities in the forward direction along the ray can be hit. Hits_wanted indicates the maximum number of hits to return. To return all hits, specify hits_wanted as zero. The list of hits is created on the heap and it is the responsibility of the caller to delete the hits in this list.

When several connected entities are wanted, e.g., all edges of a face, pick one entity, for example, the face, and the others are found by following the model pointers.

If the ray hits the interior of a face at a point at least the length of the ray radius from any edge or vertex of the face, it returns the face. If the ray hits an edge of a face or passes within ray radius of the edge, it returns the edge. If the ray passes within ray radius of a vertex, it returns the vertex.

To pick edges or vertices, it is often helpful to increase the ray radius. To pick a face, keep the ray radius small to avoid picking edges or vertices of the face. If the ray lies in the surface (planar or ruled) of a face and crosses the interiors of the face, the edges or vertices of the face are returned.

A call to this API will cause boxes to be computed, causing a model change and creating a bulletin board. To make the process Read-only, call api_note_state before the call to api_raytest_body, api_note_state, api_change_state, and api_delete_ds after the call to api_raytest_body.

Errors: Pointer to body is NULL or not to a BODY. Zero length ray_direction specified. Ray radius less than SPAresabs.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_raytest_ents

Function: Object Relationships, Ray Testing

Action: Gets the list of hits when a ray is fired at a one or more entities.

Prototype:

```
outcome api_raytest_ents (
    SPAposition const& ray_point,// starting point of
                             // ray
    SPAunit_vector const&        // direction of
        ray_direction,           // the ray
    double ray_radius,           // radius of ray
    int hits_wanted,             // number of hits
                                 // requested
    int n_target_ents,           // number of target
                                 // entities
    ENTITY* target_ents[],       // array of target
                                 // entities
    hit*& hit_list,              // hits along the ray
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”intersct/kerndata/raytest/raytest.hxx”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/unitvec.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API fires a ray at an array of entities from the given ray point in the given ray direction with the given ray radius. It returns a list that contains the n hits nearest the ray point (where n is the number of hits recorded.) Only entities in the forward direction along the ray can be hit. Hits_wanted indicates the maximum number of hits to return. To return all hits, specify hits_wanted as zero. The caller is responsible for deleting the list of hits.

When several connected entities are wanted; e.g., all edges of a face, pick one entity, the face, and the others are found by following model pointers.

A face–ray intersection counts whether the ray enters or leaves the face. If the ray hits the interior of a face at a point at least the length of the ray radius from any edge or vertex of the face, it returns the face. If it hits an edge of a face or passes within ray radius of the edge, it returns the edge. If it passes within ray radius of a vertex, it returns the vertex.

To pick edges or vertices, it is often helpful to increase the ray radius. To pick a face, keep the ray radius small to avoid picking edges or vertices of the face. If the ray lies in the (planar or ruled) surface of a face, and crosses the interior of the face, the edges or vertices of the face cut by the ray are returned.

This API fires a ray at a subset of the entities in a body; e.g., at a set of faces already selected in some way. Ray tests applied to other entities (wires, lumps, shells, subshells, faces, edges, or vertices) given directly in the array of entities, are made in the local space of the entity.

A call to this API will cause boxes to be computed, causing a change and the creation of a bulletin board. To make the process Read–only, call api_note_state before the call to api_raytest_ents.

Errors: Zero length ray_direction specified. Ray radius less than SPAresabs. Entity pointer in array is NULL.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_ray_test_body

Function: Object Relationships, Ray Testing

Action: Gets the list of entities that are hit when a ray is fired at a body.

Prototype:

```
outcome api_ray_test_body (
    SPAposition const& ray_point,// starting point of
                                 // ray
    SPAunit_vector const&        // direction of
        ray_direction,           // the ray
    double ray_radius,           // radius of ray
    int hits_wanted,             // number of hits
                                 // requested
    BODY* target_body,           // target body
    ENTITY_LIST& entities_hit,   // list of entities
                                 // hit by ray
                                 // returned
    double*& ray_parameters,     // parameters of ray
                                 // returned
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/unitvec.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API fires a ray at a body from the given ray point in the given ray direction with given ray radius. It returns a list that contains the *n* entities nearest the ray point (where *n* is the number of hits recorded) and an array that holds the *n* parameter values of the points along the ray. Only entities in the forward direction along the ray can be hit. hits_wanted indicates the maximum number of hits to return. To return all hits, specify hits_wanted as zero.

If ray_parameters is not NULL, it must point to an array of doubles large enough to hold all of the returned parameters. When ray_parameters is specified as NULL, an array large enough to hold the returned parameters is allocated and its address is returned. The caller is responsible for deleting the array.

When several connected entities are wanted; e.g., all edges of a face, pick one entity the face and the others are found by following the model pointers.

If the ray hits the interior of a face at a point at least the length of the ray radius from any edge or vertex of the face, it returns the face. If it hits an edge of a face or passes within ray radius of the edge, it returns the edge. If it passes within ray radius of a vertex, it returns the vertex.

To pick edges or vertices, it is often helpful to increase the ray radius, but to pick a face, keep the ray radius small to avoid picking edges or vertices of the face. If the ray lies in the surface (planar or ruled) of a face and crosses the interior of the face, the edges or vertices of the face are returned. If the ray hits an entity more than once, only the hit at the smallest ray parameter is returned.

A call to this API will cause boxes to be computed, causing a change and creating a bulletin board. To make the process Read-only, call api_note_state before the call to api_ray_test_body.

Errors: Pointer to body is NULL or not to a BODY. Zero length ray_direction specified. Ray radius less than SPAresabs.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_ray_test_ents

Function: Object Relationships, Ray Testing

Action: Gets the list of entities that are hit when a ray is fired at one or more entities.

Prototype:

```
outcome api_ray_test_ents (
    SPAposition const& ray_point,// starting point of
                                 // ray
    SPAunit_vector const&        // direction of
        ray_direction,           // the ray
    double ray_radius,           // radius of ray
    int hits_wanted,             // number of hits
                                 // requested
    int n_target_ents,           // number of target
                                 // entities
    ENTITY* target_ents[],       // array of target
                                 // entities
    ENTITY_LIST& entities_hit,   // list of entities
                                 // hit returned
    double*& ray_parameters,     // parameters of the
                                 // ray returned
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/unitvec.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API fires a ray at an array of entities from the given ray point in the given ray direction with the given ray radius. It returns a list that contains the *n* entities nearest the ray point (where *n* is the number of hits recorded) and an array that holds the *n* parameter values of the points along the ray. Only entities in the forward direction along the ray can be hit. hits_wanted indicates the maximum number of hits to return. To return all hits, specify hits_wanted as zero.

If ray_parameters is not NULL, it must point to an array of doubles large enough to hold all of the returned parameters. When ray_parameters is specified as NULL, an array large enough to hold the returned parameters is allocated and its address is returned. The caller is responsible for deleting the array. When several connected entities are wanted; e.g., all edges of a face, pick one entity the face and the others are found by following the model pointers.

A face–ray intersection counts whether the ray enters or leaves the face. If the ray hits the interior of a face at a point at least the length of the ray radius from any edge or vertex of the face, it returns the face. If it hits an edge of a face or passes within ray radius of the edge, it returns the edge. If it passes within ray radius of a vertex, it returns the vertex.

To pick edges or vertices, it is often helpful to increase the ray radius, but to pick a face, keep the ray radius small to avoid picking edges or vertices of the face. If the ray lies in the surface (planar or ruled) of a face and crosses the interior of the face, the edges or vertices of the face are returned.

If the ray hits an entity more than once, only the hit at the smallest ray parameter is returned. A ray test applied to a body is made in the global space of the body, and tests all faces, edges, and vertices of the lumps and wires of the body in global body space.

This API fires a ray at a subset of the entities in a body; e.g., at a set of faces already selected in some way. Ray tests applied to other entities (wires, lumps, shells, subshells, faces, edges, or vertices) given directly in the array of entities are made in the local space of the entity.

A call to this API will cause bounding boxes to be computed, causing a model change and creating a bulletin board. To make the process read–only, call api_note_state before the call to api_ray_test_ents.

Errors: Zero length ray_direction specified. Ray radius less than SPAresabs. Entity pointer in array is NULL.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_set_entity_pattern

Function: Patterns

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

Prototype:

```
outcome api_set_entity_pattern (
    ENTITY* in_ent,              // seed entity
    pattern* in_pat,             // pattern to apply
    logical copy_pat             // copy pattern and
         = TRUE,                 // apply the copy
                                 // instead of in_pat
    int seed_index               // pattern index for
        = 0,                     // seed entity
    ENTITY_LIST& no_cross_faces// list of faces that
        =*(ENTITY_LIST*)NULL_REF,// bound the seed
    PAT_CHECK_TYPE check     // desired checking
        = PAT_DONT_CHECK,    // option
    AcisOptions* ao = NULL   // ACIS options
    );
```

```
outcome api_set_entity_pattern (
    ENTITY_LIST in_ents,     // list of seed entities
    pattern* in_pat,         // pattern to apply
    logical copy_pat         // copy pattern and apply
         = TRUE,             // the copy
                             // instead of in_pat
    int seed_index           // pattern index for
        = 0,                 // seed entity
    ENTITY_LIST& no_cross_faces// list of faces that
        =*(ENTITY_LIST*)NULL_REF,// bound the seed
    PAT_CHECK_TYPE check     // desired checking
        = PAT_DONT_CHECK,    // option
    AcisOptions* ao = NULL   // ACIS options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "intersct/kernapi/api/intrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernutil/law/pattern.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Applies the pattern in_pat to the seed entity in_ent or list of seed entities in_ents. By default, a copy of the pattern is made, and it is the copy that is actually applied to the entity. This behavior can be overridden by setting copy_pat to FALSE. However, when copying is overridden and in_pat is shared by multiple bodies, a transform placed upon the bodies are transferred to the pattern multiple times, behavior that is clearly undesirable. Also by default, the seed entity is associated with the first pattern element (index 0), but may be associated another element by furnishing the associated zero–based seed_index.

For cases in which the pattern is applied to a "bump" on a substrate rather than to an autonomous entity, the limits of the bump are automatically computed, but the user may choose to override the default limits by furnishing a list of no_cross_faces.

For performance reasons, the function does not check the generated pattern of entities for intersection, containment, or compatibility unless the user changes the checking option check from its default value. This argument takes the following values:

PAT_DONT_CHECK —– do no checking

PAT_CHECK_DONT_FIX —– check the patterned entities and roll back in the case of failure

PAT_CHECK_AND_FIX —– check the patterned entities. If only the containment check fails, drop the problem elements from the pattern and re–apply it to the seed. If either the intersection or compatibility check fails, roll back.

Errors: An entity type not supporting patterns is specified, or (if checking is enabled through the check option) the pattern has problems with intersection, containment or compatibility.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_silhouette_edges

Function: Silhouette and Isoparametric Curves

Action: Gets a list of silhouette edges for a given face.

Prototype:
```
outcome api_silhouette_edges (
    FACE* face,                     // face to be
                                    // examined
    const SPAtransf& ftrans,        // transform
    const SPAposition& from_point,// position of
                                    //viewer
    const SPAposition& to_point,// position toward
                                    // which the viewer
                                    // looks
    int projection_type,            // (0 = parallel
                                    // 1 = perspective)
    ENTITY_LIST* edgelist,          // list of silhouette
                                    // edges returned
    AcisOptions* ao = NULL          // ACIS options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/transf.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API computes the silhouette curves on the surface of the face and trims them to the boundaries of the face.

The silhouette curves can be calculated for either a parallel or perspective projection.

It retrieves a list of EDGEs representing the bounded portion of the silhouette curves within the face.

Errors: Pointer to face is NULL or not to a FACE.
From and to points are coincident.

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: Changes model

api_terminate_intersectors

Function: Modeler Control, Intersectors

Action: Terminates the intersector library.

Prototype:
```
outcome api_terminate_intersectors ();
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kernapi/api/intrapi.hxx

Effect: System routine

bs3_curve_ana_int

Function: Spline Interface, Intersectors

Action: Intersects a spline with a general (analytic) surface.

Prototype:
```
curve_surf_int* bs3_curve_ana_int (
    bs3_curve cur,            // bs3_curve
    surface const& sur,       // surface
    double fitol,             // tolerance
    box const&                // region of
        region_of_interest    // interest
);
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”intersct/spline/bs3_crv/bs3cutil.hxx”
#include ”baseutil/vector/box.hxx”
#include ”kernel/kerngeom/surface/surdef.hxx”
#include ”kernel/kernint/intcusf/cusfint.hxx”
#include ”kernel/spline/bs3_crv/bs3curve.hxx”
```

Description:	Intersects a spline with a general (analytic) surface. This should not be used for a spline surface.
Errors:	None
Limitations:	None
Library:	intersct
Filename:	intr/intersct/spline/bs3_crv/bs3cutil.hxx
Effect:	Changes Model

bs3_curve_con_int

Function: Spline Interface, Intersectors

Action: Intersects the given parametric curve with a cone.

Prototype:

```
curve_surf_int* bs3_curve_con_int (
    bs3_curve bs,                // given curve
    cone const& con,             // given cone
    double fitol,                // fit tolerance
    box const&                   // region of
        region_of_interest       // interest
);
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/spline/bs3_crv/bs3cutil.hxx”
#include ”baseutil/vector/box.hxx”
#include ”kernel/kerngeom/surface/condef.hxx”
#include ”kernel/kernint/intcusf/cusfint.hxx”
#include ”kernel/spline/bs3_crv/bs3curve.hxx”
```

Description:	In all other respects, the definition of this routine is the same as that of bs3_curve_pla_int.
Errors:	None
Limitations:	None
Library:	intersct
Filename:	intr/intersct/spline/bs3_crv/bs3cutil.hxx
Effect:	Changes Model

bs3_curve_pla_int

Function: Spline Interface, Intersectors

Action: Intersects the given parametric curve with a plane.

Prototype:

```
curve_surf_int* bs3_curve_pla_int (
    bs3_curve bs,                 // given curve
    plane const& pla,             // given plane
    double fitol,                 // fit tolerance
    box const&                    // region of
        region_of_interest        // interest
);
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/spline/bs3_crv/bs3cutil.hxx”
#include ”baseutil/vector/box.hxx”
#include ”kernel/kerngeom/surface/pladef.hxx”
#include ”kernel/kernint/intcusf/cusfint.hxx”
#include ”kernel/spline/bs3_crv/bs3curve.hxx”
```

Description: Intersects the given parametric curve with a plane and returns an ordered list of intersection points together with information about relationships between the curve and surface in the neighborhood of each intersection point.

Tangency and extended regions of coincidence are related to the given tolerance, and not standard system tolerance, except when the given tolerance is less than standard.

The information returned for each intersection point is described fully in the documentation of ACIS curve-surface intersections, but a brief description follows:

int_point, the intersection point, is an object-space position. At a tangency, where the “true” intersection point is not well-defined, this point will be the average of the points of minimum (or maximum if the curve penetrates the surface) distance. For a region of coincidence, the ends of that region are given, as two intersections.

param, the parameter value of int_point on the curve.

surf_param, if the surface is parametric, this is the parameter values on the surface of int_point. If not, it is undefined.

low_rel, high_rel, is the relationship between the curve and the surface in the neighborhood of the intersection, on the side with the lower curve parameter and the side with higher curve parameter. Classifications are in and out for normal crossings, in-tangent and out-tangent (for tangency) and coincident.

fuzzy, low_param, high_param, if the region of uncertainty in the intersection is significantly higher than tolerance (at a tangency or very low-angle crossing) the range of curve parameter values over which the curve is entirely within tolerance of the surface are given as (low_param, high_param), and fuzzy is set TRUE. At well-defined intersections, fuzzy is FALSE and both low_param and high_param will be equal to param.

Intersection points outside the region of interest need not be evaluated, but do no harm to ACIS if they are.

A special case arises if the curve is everywhere coincident with the surface (within the region of interest). If so, a single intersection point will be returned at an arbitrary position on the curve, with low_rel and high_rel both set to coincident.

ACIS makes an assumption for tangent intersections that low_param and high_param are symmetric about param.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/spline/bs3_crv/bs3cutil.hxx

Effect: Changes Model

bs3_curve_project

Function: Spline Interface, Intersectors

Action: Projects the given bs3_curve along the given path onto the given plane.

Prototype:

```
bs3_curve bs3_curve_project (
    bs3_curve in_cur,        // given curve
    const curve* path_c,     // path curve along which
                             // in_cur is projected
    const plane& to_surf     // plane onto which
                             // in_cur is projected
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/spline/bs3_crv/bs3cutil.hxx”
#include ”kernel/kerngeom/curve/curdef.hxx”
#include ”kernel/kerngeom/surface/pladef.hxx”
#include ”kernel/spline/bs3_crv/bs3curve.hxx”
```

Description:	This routine takes the input curve and projects the control polygon onto the plane along the given curve and creates the control polygon for the result.
Errors:	None
Limitations:	None
Library:	intersct
Filename:	intr/intersct/spline/bs3_crv/bs3cutil.hxx
Effect:	Changes Model

bs3_curve_sph_int

Function: Spline Interface, Intersectors

Action: Intersects the given parametric curve with a sphere.

Prototype:

```
curve_surf_int* bs3_curve_sph_int (
    bs3_curve bs,                 // given curve
    sphere const& sph,            // given sphere
    double fitol,                 // fit tolerance
    box const&                    // region of
        region_of_interest        // interest
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/spline/bs3_crv/bs3cutil.hxx”
#include ”baseutil/vector/box.hxx”
#include ”kernel/kerngeom/surface/sphdef.hxx”
#include ”kernel/kernint/intcusf/cusfint.hxx”
#include ”kernel/spline/bs3_crv/bs3curve.hxx”
```

Description:	In all other respects, the definition of this routine is the same as that of bs3_curve_pla_int.
Errors:	None
Limitations:	None
Library:	intersct
Filename:	intr/intersct/spline/bs3_crv/bs3cutil.hxx
Effect:	Changes Model

bs3_curve_spl_int

Function: Spline Interface, Intersectors

Action: Intersects the given parametric curve with a spline surface.

Prototype:

```
curve_surf_int* bs3_curve_spl_int (
    bs3_curve bs_cur,        // given curve
    spline const& spl,       // given spline surface
    double fitol,            // fit tolerance
    box const& b             // region of interest
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/spline/bs3_crv/bs3cutil.hxx”
#include ”baseutil/vector/box.hxx”
#include ”kernel/kerngeom/surface/spldef.hxx”
#include ”kernel/kernint/intcusf/cusfint.hxx”
#include ”kernel/spline/bs3_crv/bs3curve.hxx”
```

Description: In all other respects, the definition of this routine is the same as that of bs3_curve_pla_int.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/spline/bs3_crv/bs3cutil.hxx

Effect: Changes Model

bs3_curve_tor_int

Function: Spline Interface, Intersectors

Action: Intersects the given parametric curve with a torus.

Prototype:

```
curve_surf_int* bs3_curve_tor_int (
    bs3_curve bs,                // given curve
    torus const& tor,            // given torus
    double fitol,                // fit tolerance
    box const&                   // region of
        region_of_interest       // interest
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/spline/bs3_crv/bs3cutil.hxx”
#include ”baseutil/vector/box.hxx”
#include ”kernel/kerngeom/surface/tordef.hxx”
#include ”kernel/kernint/intcusf/cusfint.hxx”
#include ”kernel/spline/bs3_crv/bs3curve.hxx”
```

Description: In all other respects, the definition of this routine is the same as that of bs3_curve_pla_int.

The standard version of this function as supplied is not dependent on the underlying surface package, in that it uses only ACIS kernel facilities together with other bs3_curve functions. Re–implementation is optional for a different surface package.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/spline/bs3_crv/bs3cutil.hxx

Effect: Changes Model

d3_sf_sf_int

Function: Intersectors

Action: Intersects two surfaces.

Prototype:

```
surf_surf_int* d3_sf_sf_int (
    const surface& sf0,      // first surface
    const surface& sf1,      // second surface
    const box& b_in,         // region of interest
    double fitol             // fit tolerance for
        = resfit,            // returned intcurves
    const help_point* help_pts// List of entry, exit,
        = 0,                 // help, isolated points,
                             // and terminators
    COMPLETENESS_TYPE        // Specifies the
        completeness         // completeness of the
        = ALL_CURVES,        // results required
    logical safe_area        // TRUE if all
        = FALSE,             // terminators, turning
                             // points and isolated
                             // points have been
                             // supplied, or none
                             // exist
    const double off0        // offset of first
        = 0.0,               // surface
    const double off1        // offset of second
        = 0.0,               // surface
    logical abort_on_illegal_surf //
        = FALSE              );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kernint/d3_ssi/ssi_inf.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/box.hxx”
#include ”kernel/kerngeom/surface/surdef.hxx”
#include ”kernel/kernint/intsfsf/sfsfint.hxx”
#include ”kernel/kernutil/d3_fn2/fn2.hxx”
```

Description: This surface intersection function takes two surfaces and a box, and finds all intersection curves, isolated intersection points and coincident regions. The intersections are returned as surf_surf_ints.

Although the algorithm can work on any types of surface, it is designed for use on parametric surfaces with boundaries, and treats all surfaces as though they have boundaries or are periodic. The boundaries of an implicit surface in a nonperiodic direction are parameter curves at the limit of the par_box returned by the surface param_range function, given the intersection box.

The full algorithm finds all intersections on the surface boundaries first to obtain points to march from, and generate intersection curves. It also performs subdivision, to find internal points from which to march. The application can request that either the boundary processing stage or the subdivision (or both) be missed, so that only the marching and creation of result curves remains. In this case, the application must supply the missing information, by providing help points (see below).

Three flags are supplied to d3_sf_sf_int, to control what the algorithm does and what data it should expect:

int completeness

This flag controls the number of interaction curves that are returned. It may have one of five values:

completeness = 1 – Only a single intersection curve is required, and a help point of some sort has been supplied for it.

completeness = 2 – All curves corresponding to input data will be found. Again, help point(s) must be supplied or no results will be returned. Curves will only be returned if they lie on one or more of the help points, or if they are connected to such a curve through a terminator (so, for example, both parts of a figure–of–eight curve will be returned if a single help point on one part is supplied). This is an important difference between completeness = 1 and completeness = 2.

completeness = 3 – Curves that lie on one or more of the supplied help points, or that are connected to such a curve through a terminator are returned, and also any curves that are found by the boundary processing.

completeness = 4 – Curves that lie on one or more of the supplied help points, or that are connected to such a curve through a terminator are returned, and also any curves that are found by the subdivision.

completeness = 5 (the default) – All curves will be found, using the full algorithm. There is no need to supply any help point information.

Note that using completeness = 2 will find all possible curves if the input data is complete, in the sense that the following help point data is supplied:

1. All entry points (points at which an intersection curve enters any boundary of either surface).
2. All exit points (points at which an intersection curve leaves any boundary of either surface).
3. A help point (including a terminator) on each intersection curve which does not reach the boundary.
4. All terminators which happen to be on any boundary of either surface.
5. All isolated points which happen to be on any boundary of either surface. This includes tangent isolated points, at which the surfaces just touch, or points that are isolated simply because they are on the boundary.

If isolated points are not provided then they will not be included in the results structure, but there is no other adverse effect.

logical safe_area

If this flag is true, then it indicates that all terminators, turning points and isolated points have been supplied. It does not affect the results returned by the algorithm but enables it to work much faster.

If incomplete results are sufficient (for example only a single curve has been requested) then it is only necessary to supply the turning points, isolated points and terminators on or near these results.

A tangent curve starts and ends at terminators, not entry and exit points. Also, a help point on a tangent curve should be supplied as a terminator.

The various entry, exit, isolated and help points and terminators that might be supplied to d3_sf_sf_int are collectively called help_points. The following is a classification for the help points.

```
enum help_point_type
    {
    Entry,      // A point on the surface boundary
                // at which an
                // intersection curve enters
    Exit,       // A point on the surface boundary at
                // which an intersection curve leaves
    Help,       // A point on the intersection which
                // is not an entry, exit or isolated
                // point or a terminator
    Terminator, // A point on both surfaces at which
                // the surfaces are tangent or
                // singular, which is not simply an
                // isolated point of contact
    Isolated,   // An isolated point of contact on
                // both surfaces
    Flat,       // Either a terminator or isolated
                // point.
    Turning_point  // A pair of points, one on each
                // surface, at which the surface
                // normals are parallel or
                // antiparallel
    };
```

Except for the Flat type (which indicates either a terminator or an isolated point), these categories do not overlap and should not be confused.

For example, classifying a point as Help when it should be an entry point will lead to errors in the intersection algorithm.

Help points are supplied to d3_sf_sf_int in a linked list of the following structures.

```
class DECL_INTR help_point ACIS_BASE_CLASS {
    public:
    position P;
    par_pos uv0;    // Parameters on first surface;
                    // set iff the surface is
                    // parametric
    par_pos uv1;    // Parameters on second surface;
                    // set iff the surface is
                    // parametric
    help_point_type type;
    logical on_boundary;// TRUE if the help point
                    // is known to be on a boundary of
                    // either surface.
    help_point* next;

    help_point(
        const position& aP,
        const par_pos& auv0,
        const par_pos& auv1,
        help_point_type atype,
        logical aon_boundary,
        help_point* anext);
    };
```

Note that on_boundary should always be TRUE if the type is an entry or exit point.

The algorithm can also be used on offset surfaces, defined by supplying the offsets specifically. However, help points are required and complete results cannot be guaranteed in this case so the completeness flag cannot be set to 5.

Errors:	None
Limitations:	None
Library:	intersct
Filename:	intr/intersct/kernint/d3_ssi/ssi_inf.hxx
Effect:	Changes Model

delete_hit_list

Function: Ray Testing, Debugging

Action: Deletes a hit list.

Prototype:
```
void delete_hit_list (
    hit* list                // hit list to delete
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”intersct/kerndata/raytest/raytest.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/kerndata/raytest/raytest.hxx

Effect: System routine

find_face_face_distance

Function: Object Relationships

Action: Calculates the distance between two faces.

Prototype:
```
double find_face_face_distance (
    FACE* face1,             // face 1
    FACE* face2,             // face 2
    SPAposition& p1,         // closest point on ent 1
    SPAposition& p2,         // closest point on ent 2
    param_info& ent1_info    // entity 1 param info
        =*(param_info*)NULL_REF,
    param_info& ent2_info    // entity 2 param info
        =*(param_info*)NULL_REF
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”intersct/sg_husk/query/sgquery.hxx”
#include ”intersct/kernapi/api/intrapi.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kerndata/top/face.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: intersct

Filename: intr/intersct/sg_husk/query/sgquery.hxx

Effect: Read–only

raytest_edge

Function: Ray Testing

Action: Tests a ray against the given edge.

Prototype:
```
hit* raytest_edge (
    ray& this_ray,           // ray
    EDGE* this_edge          // target edge
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”intersct/kerndata/raytest/raytest.hxx”
#include ”kernel/kerndata/top/edge.hxx”
```

Description: The routine raytest fires a ray at a body and returns on the heap a list of hits or NULL if no hits are found. The heap contains the entity hit, the type of hit, and the parameter value at the hit. A value of zero for hits permitted means all hits are returned. Logicals allow shells or wires or both to be selected as targets for the ray. Only entities in the forward direction along the ray can be hit.

If the ray hits the interior of a face, the face is returned; if it hits an edge of a face (i.e., two adjacent faces), the edge is returned; and if it hits all faces adjacent at a vertex, the vertex is returned.

Errors: None

Limitations: A call to this routine may cause boxes to be computed and thus the model may change.

Library: intersct

Filename: intr/intersct/kerndata/raytest/raytest.hxx

Effect: Read–only

raytest_face

Function: Ray Testing

Action: Tests a ray against the given face.

Prototype:

```
hit* raytest_face (
    ray& this_ray,           // ray
    FACE* this_face          // target face
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/kerndata/raytest/raytest.hxx”
#include ”kernel/kerndata/top/face.hxx”
```

Description: The routine raytest fires a ray at a body and returns on the heap a list of hits or NULL if no hits are found. The heap contains the entity hit, the type of hit, and the parameter value at the hit. A value of zero for hits permitted means all hits are returned. Logicals allow shells or wires or both to be selected as targets for the ray. Only entities in the forward direction along the ray can be hit.

If the ray hits the interior of a face, the face is returned; if it hits an edge of a face (i.e., two adjacent faces), the edge is returned; and if it hits all faces adjacent at a vertex, the vertex is returned.

Errors: None

Limitations: A call to this routine may cause boxes to be computed and thus the model may change.

Library: intersct

Filename: intr/intersct/kerndata/raytest/raytest.hxx

Effect: Read–only

sg_get_transform

Function: Transforms

Action: Returns the transformation of the parent body of the the provided entity.

Prototype:

```
SPAtransf& sg_get_transform (
    ENTITY* entity           // entity starting point
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/sg_husk/query/sgquery.hxx”
#include ”baseutil/vector/transf.hxx”
#include ”kernel/kerndata/data/entity.hxx”
```

Description: Routine which obtains the transformation of the parent body of the following entities: body, lump, shell, subshell, face, loop, coedge, edge, vertex, and wire.

Errors:	None
Limitations:	None
Library:	intersct
Filename:	intr/intersct/sg_husk/query/sgquery.hxx
Effect:	Read–only

sg_split_edge_at_convexity

Function: Intersectors

Action: Splits edge convexity at convexity points.

Prototype:

```
void sg_split_edge_at_convexity (
    EDGE* edge               // edge to be split
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”intersct/sg_husk/split/esplit.hxx”
#include ”kernel/kerndata/top/edge.hxx”
```

Description: ACIS does not allow edges to change convexity over the course of their length. Changes in edge convexity are reported by the entity checker at level 60. ACIS itself should never create such edges, but it is possible that models translated from other systems may have edge convexity errors. sg_split_edge_at_convextiy finds the places along the edge where the convexity changes and inserts vertices at those spots.

Note *This function does not handle tolerant edges.*

Errors:	None
Limitations:	None
Library:	intersct
Filename:	intr/intersct/sg_husk/split/esplit.hxx
Effect:	Changes model

sg_split_edge_at_vertex

Function: Intersectors

Action: Splits edge at vertex which is know to lie within the domain of the edge.

Prototype:

```
void sg_split_edge_at_vertex (
    EDGE* edge,              // edge to be split
    VERTEX* new_vertex,      // vertex known to lie in
                             // interior of edge
    ENTITY_LIST& coedge_list=   // returned list of
        *(ENTITY_LIST*)NULL_REF, // coedges
    logical split_geometry= // hard split on geometry
        FALSE                //
    );

void sg_split_edge_at_vertex (
    EDGE* old_edge,          // edge to be split
    VERTEX* new_vertex,      // vertex known to lie in
                             // interior of edge
    double vert_param,       // parameter at which the
                             // vertex lies
    ENTITY_LIST& coedge_list=   // returned list of
        *(ENTITY_LIST*)NULL_REF, // coedges
    logical split_geometry= // hard split on geometry
        FALSE                //
    );
```

Includes:

```
#include "baseutil/debug/module.hxx"
#include "intersct/sg_husk/query/sgquery.hxx"
#include "intersct/sg_husk/split/spled.err"
#include "kernel/geomhusk/get_top.hxx"
#include "kernel/kerndata/attrib/attrib.hxx"
#include "kernel/kerndata/geom/cnstruct.hxx"
#include "kernel/kerndata/geom/curve.hxx"
#include "kernel/kerndata/geom/point.hxx"
#include "kernel/kerndata/top/alltop.hxx"
#include "kernel/kerngeom/curve/intdef.hxx"
#include "kernel/kerngeom/d3_crv/edge_cnx.hxx"
#include "kernel/sg_husk/query/q_vert.hxx"
```

Description:

Splits an edge at a vertex which is known to lie in the interior of the edge; the old edge will go from the start to the new while the new edge will go from the new to the end.

If vert_param is not provided, it will calculate this parameter from the position of the new vertex with respect to the edge.

All coedges sharing the edge are also split with the new coedge sharing the new edge and the partner order of coedges around the old edge corresponding to the partner order of the new coedges around the new edge.

If the split_geometry option is set to TRUE, the curve underneat is split if the parameter lies on a discontinuity, otherwise, the new edges share the same curve.

Errors:	None
Limitations:	None
Library:	intersct
Filename:	intr/intersct/sg_husk/split/esplit.hxx
Effect:	Changes model