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_blend_edges

Function: Blending, Modifying Models

Action: Blends edges of a solid using round blends.

Prototype:

```
outcome api_blend_edges (
    ENTITY_LIST& edges,      // list of edges to blend
    double radius,           // blend radius
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API creates round blends along one or more edges based on the given blend radius. The blend radius is specified in the local space of the body. If all edges that meet at a specific vertex are in the entity list, the vertex is also blended. If a vertex is marked as rounded, the details of the blend face are found from the radius of the edges that end in the vertex.

Errors:	None
Limitations:	None
Library:	blend
Filename:	blnd/blend/kernapi/api/blendapi.hxx
Effect:	Changes model

api_blend_graph

Function: Blending, Modifying Models

Action: Locates a graph of edges.

Prototype:

```
outcome api_blend_graph (
    ENTITY* this_ent,       // an edge or vertex in
                            // the graph
    ENTITY_LIST& ent_list,  // returned list of
                            // connected entities
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.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: When given an edge or vertex, this api finds a graph of edges that meet smoothly; i.e., with similar blend type, convexity, and radius, and with tangent continuity at unblended vertices, or edges that meet at blended vertices with the same or different blend radii or convexity. It returns the blended edges and vertices in an entity list.

Errors:	The pointer to an entity is NULL.
Limitations:	None
Library:	blend
Filename:	blnd/blend/kernapi/api/blendapi.hxx
Effect:	Read-only

api_blend_seq

Function: Blending, Modifying Models

Action: Locates a sequence of connected edges that meet smoothly.

Prototype:
```
outcome api_blend_seq (
    EDGE* this_edge,          // an edge in the
                              // sequence
    ENTITY_LIST& ents,        // returned new list of
                              // connected entities
    AcisOptions* ao           // ACIS
        = NULL                // options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Given an edge, this API finds a sequence of connected edges that meet smoothly; i.e., with tangent continuity or meet at vertices that have the same implicit blend as the edges (when the vertex is also entered into the list).

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

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Read-only

api_chamfer_edges

Function: Blending, Modifying Models

Action: Chamfers the edges of a solid.

Prototype:
```
outcome api_chamfer_edges (
    ENTITY_LIST& edges,       // edges to chamfer
    double left_range,        // left range of chamfer
    double right_range        // if negative, it’s
        = –1,                 // equal to left_range
    AcisOptions* ao           // ACIS
        = NULL                // options
    );
```

Includes:	`#include ”kernel/acis.hxx”`
	`#include ”blend/kernapi/api/blendapi.hxx”`
	`#include ”kernel/kernapi/api/api.hxx”`
	`#include ”kernel/kerndata/lists/lists.hxx”`
	`#include ”kernel/kernapi/api/acis_options.hxx”`

Description:	This API creates flat (planar) chamfers on the given edges with the given left and right ranges, where left and right are with respect to the edge direction. Ranges are always positive; if the given right range is negative, a range equal to the left range is used. For a chamfer with equal left and right ranges, the points where the chamfer surfaces meet the faces of the chamfered edge are the same as those for a round with radius equal to the chamfer range. If the ranges for left and right differ, the meeting points on the left face are the same as those for a round with radius equal to the left range, and the meeting points on the right face are the same as those for a round with radius equal to the right range.
Errors:	None
Limitations:	None
Library:	blend
Filename:	blnd/blend/kernapi/api/blendapi.hxx
Effect:	Changes model

api_chamfer_vertex

Function: Blending

Action: Creates a planar chamfer on a vertex of a solid body.

Prototype:

```
outcome api_chamfer_vertex (
    VERTEX* vertex,           // vertex to chamfer
    double offset1,           // offset along 1st edge
    EDGE* edge1               // first edge
        = NULL,
    double offset2            // offset along 2nd edge
        = –1,
    EDGE* edge2               // second edge
        = NULL,
    double offset3            // offset along 3rd edge
        = –1,
    EDGE* edge3               // third edge
        = NULL,
    logical distanceStraight// measure offset along
        = TRUE,               // the straight line
    AcisOptions* ao           // ACIS
        = NULL                // options
    );

outcome api_chamfer_vertex (
    VERTEX* vertex,           // vertex to chamfer
    double offset,            // offset along
                              // the normal
    SPAvector normal,         // normal of the
                              // chamfer plane
    AcisOptions* ao           // ACIS
        = NULL                // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/vector.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kerndata/top/vertex.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API creates a planar chamfer on a vertex of a solid body.

There are two distinct forms of the API. One is when offsets are the distances along the edges, and additional edge and offset pairs are specifically defined (and normal is not defined). In the second form, the chamfer plane is constructed to be perpendicular to the normal vector and the position of the plane is determined by offsetting it from the vertex along the normal. In this case, normal must be specified and no edges or additional offsets are defined.

In both forms, offset1 can be defined as a negative value to create a concave chamfer (when material is added to form a chamfer face). Conversely, a positive offset1 value provides no information about the chamfering type. For example, a positive offset1 can be used for removing material from convex vertices and adding material to concave ones. Therefore, ACIS must detect the chamfering type. When the vertex does not have adjacent edges, which is only possible with cones and vortex tori, ACIS analyzes the face sense. For vertices with adjacent edges, convexity of the edges is used to predict the chamfering type. If such a prediction is impossible (e.g., a vertex with mixed–convexity edges) and offset1 is positive, a convex chamfer is assumed.

In the first form of this API, if the vertex is the apex of a cone, only the first two arguments are used: a pointer to the vertex and the offset value. In this case, the offset1 specifies the radius of the sphere with the center at the vertex, which intersects the cone to obtain the intersection curve. This curve is then used to find the location of the chamfer plane (the plane passes through the curve).

If any edge touches the vertex, corresponding offsets specify either the distance along the straight line between the vertex and the point on the edge (if distanceStraight is TRUE, which is its default value) or the distance along the edge (if distanceStraight is FALSE). The chamfer plane is then placed so that it passes through all these points. If the edges are straight, distanceStraight does not matter.

The minimum number of required arguments is two: only the vertex and offset have always to be specified. If the vertex is touched by several edges and only one offset is specified, equal offsets are assumed for all edges.

If there are more than three edges touching the vertex and not all of the edge pointers (edge1, edge2, and edge3) are specified in the argument list, all offsets are assumed equal to offset1 even if offset2 and offset3 were specified, and the three “best” edges are selected according to the following algorithm: first, points are placed on all edges at offset1 from the vertex, and then three such edges are selected so that they all have the same convexity (at the points location) and whose points make the triangle with the largest perimeter.

In the form of this API with the normal specified, offset1 means the offset of the chamfer plane from the vertex along the normal. This form allows for tilted chamfers on vertices with no or one adjacent edge. If the normal supplied is a zero vector, the face normal at the vertex is used.

The following code snippet illustrates a way to use this API.

```
const double R = 30.0;   // sphere radius
const double r = R/1000; // ray radius

BODY* block = NULL;
double* ray_params = NULL;

// Create a quarter-sphere
check_outcome( result = api_make_sphere( R, block )
);
BODY* tool;
check_outcome( result = api_make_cuboid ( R, R, R,
tool ) );
check_outcome( result = api_apply_transf( tool,
translate_transf( vector( R/2,R/2,R/2 ))) );
check_outcome( result = api_intersect( tool, block)
);

// Pick a vertex to chamfer
ENTITY_LIST vertices;
check_outcome( result = api_get_ents(
position(R,0,0),unit_vector(1,0,0),r,
VERTEX_TYPE,block, vertices, ray_params ) );
ACIS_DELETE [] ray_params;

// Pick edges to pass to api_chamfer_vertex
ENTITY_LIST edges;
check_outcome( result = api_get_ents(
position(R/2,0,0),unit_vector(0,0,-1),r,EDGE_TYPE,blo
ck, edges, ray_params ) );
```

```
ACIS_DELETE [] ray_params;
check_outcome( result = api_get_ents(
position(R/2,0,R/4),unit_vector(1,0,0),r,EDGE_TYPE,bl
ock, edges, ray_params ) );
ACIS_DELETE [] ray_params;
check_outcome( result = api_get_ents(
position(R/2,R/2,0),unit_vector(1,0,0),r,EDGE_TYPE,bl
ock, edges, ray_params ) );
ACIS_DELETE [] ray_params;

// Chamfer the vertex
if( vertices.count()==1  &&  edges.count()==3 )
check_outcome( result =
api_chamfer_vertex((VERTEX*)vertices[0], R/2,
(EDGE*)edges[0], R*M_PI/4, (EDGE*)edges[1], R*M_PI/4,
(EDGE*)edges[2], FALSE ) );
```

Errors: None

Limitations: If normal is not defined, the offset along any edge cannot exceed the length of that edge, so a vertex with three edges of mixed convexity may not always chamfer successfully. When normal is specified and offset1 is positive, it may fail to detect a concave vertex with edges of mixed convexity. In this case, use a negative offset1 value to explicitly specify a concave vertex. When any offset is so large the chamfer plane should intersect the faces not containing the vertex, chamfering is not performed and the “chamfer case not implemented” error message is returned.

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_chamfer_vertices

Function: Blending

Action: Creates planar chamfers on vertices of a solid body.

Prototype:

```
outcome api_chamfer_vertices (
    ENTITY_LIST& vertices,   // list of vertices
    double offset,           // chamfer offset
    logical distanceStraight// along the straight
        = TRUE,              // line or the
                             // non–straight edge
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API creates flat (planar) chamfers on the given vertices. The first argument, vertices, specifies a list of solid VERTEX entities to be chamfered. The second argument, offset, specifies the chamfer offset, equal for all edges (if any) of all vertices.

If any edge touches a vertex, the offset specifies either the distance along the straight line between the vertex and the point on the edge (if distanceStraight is TRUE, which is its default value) or the distance along the edge (if distanceStraight is FALSE). The chamfer plane is then placed so that it passes through all these points. If the edges are straight, distanceStraight does not matter.

See api_chamfer_vertex for further details. The following code snippet illustrates a way to use this API.

```
const double H = 20.0; // block size
const double C =  H  ; // primary chamfers
const double c =  H/2; // secondary chamfers

BODY* block = NULL;

// Create a cube
check_outcome( result = api_make_cuboid( 2*H,2*H,2*H,
block ) );

// Get all original vertices
ENTITY_LIST    vertices;
check_outcome( result = api_get_vertices( block,
vertices ) );

// Chamfer the original vertices
check_outcome( result = api_chamfer_vertices(
vertices, C ) );

// Get all the secondary vertices
vertices.clear();
check_outcome( result = api_get_vertices( block,
vertices ) );

// Chamfer the secondary vertices
check_outcome( result = api_chamfer_vertices(
vertices, c ) );
```

Errors: None

Limitations: The offset cannot exceed the length of any edge adjacent to a vertex, and vertices with three edges of mixed convexity may not always chamfer successfully.

When the offset is positive, it may fail to detect concave vertices with edges of mixed convexity. In this case, use a negative offset value to explicitly specify concave vertices.

When the offset is so large the chamfer plane should intersect the faces not containing the vertex, chamfering is not performed and an error message is returned.

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_complete_blends

Function: Blending, Modifying Models

Action: Creates the blend by attaching the blend sheet to the blank body using the intersection wire.

Prototype:
```
outcome api_complete_blends (
    BODY* int_graph,         // wire body already made
    BODY* sheet,             // sheet body already
                             // made
    BODY* this_body,         // body owning blended
    AcisOptions* ao          // ACIS
        = NULL               // options
                             // edges
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Phase one of blending creates a blend sheet. Phase two of blending calls the Boolean phase one to create an intersection wire. This third phase of blending calls the phase two Booleans to attach the blend sheet to the blank body using the intersection wire.

Refer to api_make_blend_sheet, phase one and api_make_blend_wire, phase two.

Errors: The pointer to a body is NULL or does not point to a BODY.

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_concl_blend_ss

Function: Blending, Modifying Models

Action: Finishes the making of a blend sheet by single steps.

Prototype:

```
outcome api_concl_blend_ss (
    blend1_data& bl1_data,   // container of
                             // blend1 data
    BODY* sheet_body,        // blend sheet
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”blend/kernbool/blend1/bl1_data.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This routine performs the final step in the first phase of blending. It makes a sheet body for a given list of blend attributes in single steps.

First api_init_blend_ss is called to initialize the process. It is given the list of blend attributes and returns a sheet body (with no faces).

Then calls are made repeatedly to api_do_one_blend_ss until the list is exhausted (the list may well grow during processing so the form of for-loop shown above should be used).

A final call to api_concl_blend_ss concludes the making of the sheet. Further calls must be made to make the blend wire and complete the blend.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_delete_blends

Function: Blending, Modifying Models

Action: Deletes an implicit blend on each of a list of edges and vertices.

Prototype:

```
outcome api_delete_blends (
    ENTITY_LIST const& ents_list,   // entities to be
                                    // unblended
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "blend/kernapi/api/blendapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Refer to Action.

Errors: An entity in list is not an BODY, FACE, EDGE, or VERTEX.

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_delete_exp_blends

Function: Blending, Modifying Models

Action: Deletes an explicit blend on each of a list of faces.

Prototype:
```
outcome api_delete_exp_blends (
    ENTITY_LIST const& ents_list,   // entities to be
                                    // unblended
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "blend/kernapi/api/blendapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Refer to Action.

Errors: An entity in list is not a FACE.

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_do_one_blend_ss

Function: Blending, Modifying Models

Action: Executes a single step in the making of a blend sheet.

Prototype:

```
outcome api_do_one_blend_ss (
    blend1_data& bl1_data,   // container of
                             // blend1_data
    int index,               // index in list of
                             // entities to be blended
    BODY* sheet_body,        // blend sheet (updated)
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”blend/kernbool/blend1/bl1_data.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This routine performs a single step in the first phase of blending. It makes a sheet body for a given blend attributes in the list. The application has the opportunity to give the user graphical feedback on the progress of the blend.

First api_init_blend_ss is called to initialize the process. It is given the initialized blend1_data object and returns an empty sheet body.

Then calls are made repeatedly to api_do_one_blend_ss until the list is exhausted. The list may well grow during processing, so a loop such as the following should be used:

```
for ( int i = 0; ; i++ ) {
    result = do_one_blended_ent(
        bl1_data, i, blend_sheet );
    if ( result <= 0 ) break;
    ...
}
```

A final call to api_concl_blend_ss ends the making of the sheet. Further calls must be made to make the blend wire and complete the blend.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_fix_blends

Function: Blending, Modifying Models

Action: Creates explicit blends by making a new face to replace each implicitly blended edge and vertex, or by computing an entity–entity blend.

Prototype:

```
outcome api_fix_blends (
    ENTITY_LIST const& ents,// list of edges and
                            // vertices, or body
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Occurs when given a list of edges and vertices marked with implicit blends, or a BODY is passed and there is an entity–entity blend attribute on the body. The entities passed should be either edges and vertices, or a body, not a mixture.

If the edges have similar blend types, the same convexity, and the same radius, all edges that meet at an unblended vertex are blended. When edges with different blend radii, convexity, and/or setback meet at a blended vertex, they are blended. Edges with different blend radii, convexity, and/or setback that meet smoothly (their adjoining faces meet at the edge with tangent plane continuity) at an unblended vertex, are blended.

Returns the blended edges and vertices of the graph in an entity list.

Errors: An entity in list is not an EDGE or VERTEX or BODY.

Limitations: Blending a single edge cannot handle blends extending beyond the adjacent face boundaries or stopped chamfers/rounds. Blending at a vertex where two edges meet cannot handle blending of edges meeting with a discontinuity.
Blending at a vertex where three edges meet if there are mixed convexities, the order in which blends are fixed is important. The edge with a different convexity must be fixed first. For example. if there are two convex blend edges and one concave, fix the single concave edge first.
Blending at a vertex where more than three edges meet, only single blends (blending of edges and not vertices) are supported.
api_fix_blends is not designed to handle edge lists that has edges from multiple bodies. If edges from multiple bodies must be blended simultaneously, the functions api_blend_edges and api_chamfer_edges can be used.
Cannot handle mixed convexities.

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_initialize_blending

Function: Blending, Modeler Control

Action: Initializes the blending library.

Prototype:
```
outcome api_initialize_blending ();
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: System routine

api_init_blend_ss

Function: Blending, Modifying Models

Action: Initializes the making of a blend sheet using single steps.

Prototype:
```
outcome api_init_blend_ss (
    blend1_data& bl1_data,   // container of
                             // blend1 data
    BODY*& sheet_body,       // returned new blended
                             // sheet body
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”blend/kernbool/blend1/bl1_data.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This routine initializes the process for carrying out the first phase of blending (make a sheet body for a given list of blend attributes) in single steps. The application has the opportunity to give the user graphical feedback on the progress of the blend.

First api_init_blend_ss is called to initialize the process. It is given the list of blend attributes and returns an empty sheet body.

Then calls are made repeatedly to api_do_one_blend_ss until the list is exhausted (the list may well grow during processing so the form of for-loop shown above should be used).

A final call to api_concl_blend_ss ends the making of the sheet. Further calls must be made to make the blend wire and complete the blend.

Errors: No blend attributes in the list

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_make_blend_cross_curve

Function: Blending

Action: Creates a cross curve of an un–fixed blend at a given v–parameter.

Prototype:
```
outcome api_make_blend_cross_curve (
    const ATTRIB_BLEND* att,// entities to be
                            // blended
    double v_param,         // parameter
    curve*& ccrv,           // new curve created
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”blend/kernbool/blending/bl_att.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerngeom/curve/curdef.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Given a blend attribute, and a v–parameter along the blend, this routine creates a “rib” curve, running from one base entity to the other, at a location along the blend determined by the v–parameter. (The v–parameter runs along the blend, and the u–parameter runs from one base surface to the other.) The v–parameter of the blend is the same as that of its defining curve, which is generally the curve of the edge being blended.

Errors: Sometimes the blend cannot evaluate a cross curve at the given parameter, for example, if the blend radius is too big.

Limitations: Currently this is only implemented for ATTRIB_VAR_BLENDs and ATTRIB_CONST_ROUNDs.

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_make_blend_sheet

Function: Blending, Modifying Models

Action: Creates a sheet body for a given list of edges, each bearing an implicit blend attribute.

Prototype:
```
outcome api_make_blend_sheet (
    ENTITY_LIST const& ent_list,// connected edges
                                // to be blended
    BODY*& sheet_body,          // returned new
                                // sheet body
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.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 first phase of blending makes a sheet body for a given list of edges, each bearing an implicit blend attribute.

The sheet made contains one or more faces for each blended entity. Attributes link the sheet to the blend body (the body owning the blended entities), thereby facilitating the next stage of blending when the sheet is combined with the blend body. The sheet faces lie on new blend surfaces or on surfaces of the blend body where these are needed to cap blends at ends of blended edges.

Where two blend faces are made for blended edges meeting non smoothly in an unblended vertex, the faces are trimmed to one another; i.e. are mitered, and the sheet faces are joined.

Refer to api_make_blend_wire, phase two and api_complete_blend, phase three.

Errors: No entities with implicit blends in the list. Implicit blend found on an entity which is not an EDGE or VERTEX.

Limitations: Blending a single edge cannot handle blends extending beyond the adjacent face boundaries or stopped chamfers/rounds. Blending at a vertex where two edges meet cannot handle blending of edges meeting with a discontinuity.

Blending at a vertex where three edges meet if there are mixed convexities, the order in which blends are fixed is important. The edge with a different convexity must be fixed first. For example. if there are two convex blend edges and one concave, fix the single concave edge first.

Blending at a vertex where more than three edges meet, only single blends (blending of edges and not vertices) are supported.

Cannot handle mixed convexities.

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_make_blend_wire

Function: Blending, Modifying Models

Action: Creates a wire body that represents the intersection of a blend sheet with the body being blended.

Prototype:

```
outcome api_make_blend_wire (
    BODY* sheet_body,       // sheet body
    BODY* blank_body,       // blank body
    BODY*& wire_body,       // returned new
                            // wire body
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: The second phase of blending makes a wire body that represents the intersection of a blend sheet with the body being blended.

Refer to api_make_blend_sheet, phase one and api_complete_blend, phase three.

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

Limitations: Blending a single edge cannot handle blends extending beyond the adjacent face boundaries or stopped chamfers/rounds. Blending at a vertex where two edges meet cannot handle blending of edges meeting with a discontinuity.

Blending at a vertex where three edges meet if there are mixed convexities, the order in which blends are fixed is important. The edge with a different convexity must be fixed first. For example. if there are two convex blend edges and one concave, fix the single concave edge first.

Blending at a vertex where more than three edges meet, only single blends (blending of edges and not vertices) are supported.

Cannot handle mixed convexities.

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_make_flat_on_faces

Function: Blending

Action: Creates planar chamfers on vertices of a solid body.

Prototype:

```
outcome api_make_flat_on_faces (
    SPAposition ptPosition, // position
    double offset,          // offset along normal
    SPAunit_vector* normal  // normal of the future
        = NULL,             // chamfer plane
    FACE** faces            // faces to chamfer
        = NULL,
    int n_faces             // number of faces
        = 1,                // in the array
    BODY* owner             // body owning the faces
        = NULL,
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/unitvec.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description:

This API creates a flat on the given faces of a solid body. The first argument, ptPosition, specifies a position in space which may lie on the faces to be modified. It must lie on the faces if faces are not given, or may not belong to any face. The offset, which may be positive or negative, determines the offset of a chamfer plane along its normal. The normal may be given as the third argument. If the normal of the chamfer plane is not specified, it is found as an outward normal of the faces being chamfered at the position nearest to ptPosition.

If the normal is not specified, it is deduced from the other supplied parameters: either faces or owner must be specified. If a body, owner, is specified, then ptPosition must lie on one or more faces of the body. A point lying on multiple faces belongs to the intersection of these faces. A list of faces to be chamfered is then constructed. The normal is computed by averaging the corresponding outward normals of each face in the faces array.

The sign of the second argument, offset, indicates whether the flat is to be made by deleting the material (negative offset) from a convex face or by adding the material to the concave face (positive offset). Note that this sign convention is different from the one used in api_chamfer_vertex.

If the intersection of the chamfer plane with the faces produces multiple intersection curves, the curve closest to the projection of the ptPosition on the chamfer plane is used.

The following code snippet demonstrates a way to use this API.

```
const double h = 3.0;   // base height
BODY* b = NULL;

// Create a plate and apply waves on top of it
check_outcome(result =
api_make_cuboid(4*M_PI,4*M_PI,2*h, b));
check_outcome(result = api_apply_transf( b,
translate_transf( vector( 1.5*M_PI, 1.5*M_PI, 0 )))
);
law* sins = NULL;
check_outcome( result = api_str_to_law(
”vec(x,y,z+sin(x)+sin(y))”, &sins) );
check_outcome( result = api_space_warp( b, sins ) );
if( sins ) sins–>remove();

BODY* t;
check_outcome( result =
api_make_cuboid(4*M_PI,4*M_PI,3*h, t));
check_outcome( result = api_apply_transf( t,
translate_transf( vector( 1.5*M_PI, 1.5*M_PI, –1.5*h
))) );check_outcome( result = api_subtract( t, b) );

// Put flats on the mountains (negative offset =>
convex)
check_outcome( result = api_make_flat_on_faces(
position(2.5*M_PI, 2.5*M_PI, h+2), –1.95,
(unit_vector*)NULL_REF, (FACE**)NULL_REF, 0, b ) );
unit_vector uv1(–.5,–.5,1);
check_outcome( result = api_make_flat_on_faces(
position(0.5*M_PI, 0.5*M_PI, h+2), –0.25, &uv1,
(FACE**)NULL_REF, 0, b ) );

double* ray_params = NULL;      ENTITY_LIST faces;
check_outcome( result = api_get_ents(
position(0,0,(h–2)/2),unit_vector(0,0,1),0.01,FACE_TY
PE,b, faces, ray_params ) );
ACIS_DELETE [] ray_params;
FACE* top_face = (FACE*) faces[0];
unit_vector uv2(0.2, 0.3, 1);
check_outcome( result = api_make_flat_on_faces(
position(1.5*M_PI, 1.5*M_PI, 1), 1, &uv2, &top_face,
1 ) );

// This last call creates a pseudo–chamfer at one of
the side
// waves. There are no vertices––we can’t use
api_chamfer_vertex.
unit_vector uv3(0.3,–2, 1);
check_outcome( result = api_make_flat_on_faces(
position(M_PI/2, –M_PI/2, h), –0.3, &uv3,
(FACE**)NULL_REF, 1, b ) );
```

Errors: When a slant flat is placed at the (concave) center of a vortex torus, the erroneous situation of too large an offset and/or tilt is not caught and a bad body is created.

Limitations: Making the maximum possible flat at the the center of a vortex torus (when the chamfer plane just touches the torus), i.e., completely filling the toroid indentation, is not possible. Instead, use api_chamfer_vertex with the central vertices of the vortex torus.

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_preview_blends

Function: Blending

Action: Creates a “preview body” containing faces which are sufficiently complete to allow a rough preview of one or more blends.

Prototype:

```
outcome api_preview_blends (
    BODY* blank_body,        // existing body which is
                             // being blended
    BODY*& sheet_body,       // existing sheet body to
                             // dump the preview faces
                             // into, or if NULL, a
                             // new body is made up
                             // and returned
    ENTITY_LIST const&       // list of blend
        blend_atts,          // attributes to be
                             // previewed, or
                             // alternatively, the
                             // entities (EDGEs etc.)
                             // being blended
    ENTITY_LIST&             // any blends which the
        cannot_preview,      // algorithm thought it
                             // should have been able
                             // to preview, but
                             // failed, are put in
                             // here
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.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: A single 4–sided face is made up for every edge blend attribute. Vertex blends, entity–entity blends etc. are ignored. However, any setbacks on an edge are respected, so the effect of setbacks on the adjacent edges can be inspected.

The body returned (or faces inserted into it) do NOT form a true valid ACIS body. Each face has a single loop of 4 coedges, with a surface which fits exactly into the face. Every surface is an exact spline and the v parameterization is ALONG the blend, matching the edge parameterization, and the u parameter runs across from 0.0 on the left face to 1.0 on the right. There is NO topological connectivity between faces, indeed they only meet “very approximately”. The surfaces may in actual fact contain self–intersections and the like, and should NEVER be handed on to intersectors, silhouette calculations, Booleans and the like. Any algorithms that process these faces MUST be able to cope with any such problems. Note that the v parameter lines are likely to be problematic for certain algorithms that expect strictly ”legal” geometry, although the u parameter lines, taken one at a time, so to speak, should be reasonable.

THE FACES RETURNED ARE VISUALIZATION AIDS ONLY

The faces only lie ”very approximately” on the faces being blended together. Just because a preview face can be constructed DOES NOT MEAN the blend itself will succeed. Actually committing valid topology and geometry to the model is several orders of magnitude harder.

The algorithms are designed for maximum speed, constructing the preview sheet should be orders of magnitude quicker than calculating the final blend. The preview faces are always dumped into the first shell of the first lump of the body, regardless.

Errors: Should not produce errors, though you may get no preview faces.

Limitations: Currently only works for round (constant and variable) blends. Any blend attributes not supported are ignored quietly.

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Read–only

api_set_const_chamfers

Function: Blending, Modifying Models

Action: Sets an implicit constant radius chamfer on each of a list of edges or vertices.

Prototype:

```
outcome api_set_const_chamfers (
    ENTITY_LIST const& entities,    // connected
                                    // entities to
                                    // be chamfered
    double left_range,              // left range
    double right_range,             // right range
    double start_setback            // starting
        = 0,                        // setback
    double end_setback              // ending setback
        = 0,
    double start_setback_diff       // start setback
        = 0,                        // difference
    double end_setback_diff         // end setback
        = 0,                        // difference
    logical start_setback_diff_set  // TRUE if start
        = TRUE,                     // setback
                                    // difference is
                                    // set
    logical end_setback_diff_set    // TRUE if end
        = TRUE,                     // setback
                                    // difference is
                                    // set
    double start_stop_ang = 0,      // for future use
    double end_stop_ang = 0,        // for future use
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API attaches implicit chamfers as attributes to each entity in the list. For a chamfer with equal left and right ranges, the points where the chamfer surfaces meets the faces of the chamfered edge are the same as those for a round with radius equal to the chamfer range. If the ranges for left and right differ, the meeting points on the left face are the same as those for a round with radius equal to the left range, and the meeting points on the right face are those for a round with radius equal to the right range. The advantage of this definition of range is that chamfers behave similarly to rounds—as the angle between the faces meeting in the chamfered edge approaches pi, the width of the chamfer face shrinks to zero.

The sense of left and right is taken from the sequence of entities in the list. Entities whose sense is reversed compared to the list have the chamfer set, but left and right ranges are interchanged. If the list contains one edge or two entities that form a closed sequence, the sense of left and right is taken from the first edge in the list. The chamfer range(s) must be supplied in the local space of the body.

Setbacks along the left and right spring curves can be made to differ and a nonorthogonal cross curve obtained by giving a nonzero (signed) value of (right setback – left setback) and setting the appropriate logical argument TRUE. At the start, the actual setbacks are found by

```
start_right_setback = start_setback +
    (0.5 * start_sb_diff)
start_left_setback = start_setback –
    (0.5 * start_sb_diff)
```

If start_sb_diff_set is given as FALSE, the setbacks are determined from the intersection of the left or right spring curves with the spring curves of the neighboring blended edges at the start. To ensure an orthogonal cross curve at the start, the setback difference should be set to zero and the logical start_sb_diff_set given as TRUE. These are the default values. The end is treated similarly.

Errors: An entity in the list is not an EDGE.
Left or right range less than –SPAresabs.
Start or end setback less than –SPAresabs.

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_set_const_rounds

Function: Blending, Modifying Models

Action: Sets an implicit constant radius round on each entity on a list of edges or vertices.

Prototype:
```
outcome api_set_const_rounds (
    ENTITY_LIST const& entities,    // entities to
                                    // be rounded
    double radius,                  // round radius
    double start_setback            // starting
        = 0,                        // setback
    double end_setback              // ending setback
        = 0,
    double start_setback_diff       // start setback
        = 0,                        // difference
    double end_setback_diff         // end setback
        = 0,                        // difference
    logical start_setback_diff_set  // TRUE if start
        = TRUE,                     // setback
                                    // difference is
                                    // set
    logical end_setback_diff_set    // TRUE if end
        = TRUE,                     // setback
                                    // difference is
                                    // set
    double start_stop_ang = 0,      // for future use
    double end_stop_ang = 0,        // for future use
    blend_how eprop                 // for future use
        = bl_how_default,           // for future use
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernbool/blending/bl_enum.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API attaches implicit rounds as attributes to each edge in the list. Supplies the round radius and start and end setbacks in the local space of the body. The start and end setbacks are zero or positive. They determine the approximate distance from the blended start or end vertex where the blend surface for the vertex ends and the blend surface for the edge begins. Setback is ignored if the vertex is not blended.

Setbacks along the left and right spring curves can be made to differ and a non circular cross curve obtained by giving a nonzero (signed) value of (right setback – left setback) and setting the appropriate logical argument TRUE. At the start, the actual setbacks are found by

```
start_right_setback = start_setback +
    (0.5 * start_sb_diff)
start_left_setback = start_setback –
    (0.5 * start_sb_diff)
```

If start_sb_diff_set is given as FALSE, the setbacks are determined from the intersection of the left or right spring curves with the spring curves of the neighboring blended edges at the start. To ensure a circular cross curve at the start, the setback difference should be set to zero and the logical start_sb_diff_set given as TRUE. These are the default values. The end is treated similarly.

Errors: An entity in the list is not an EDGE.
Radius less than –SPAresabs.
Start or end setback less than –SPAresabs.

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_set_exp_const_chamfer

Function: Blending, Modifying Models

Action: Sets an explicit constant chamfer attribute on a face.

Prototype:

```
outcome api_set_exp_const_chamfer (
    FACE* bl_face,              // face to be given
                                // attribute (not
                                // defaulted)
    FACE* left_face             // left support face
        = NULL,
    FACE* right_face            // right support face
        = NULL,
    double const& left_range    // left range
        =*(double*)NULL_REF,
    double const& right_range   // right range
        =*(double*)NULL_REF,
    logical const& cvxty        // convexity, TRUE
        =*(logical*)NULL_REF,   // for convex
    plane const& mid_plane      // approximate normal
        =*(plane*)NULL_REF,     // plane at mid-point
                                // of spine
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kerngeom/surface/pladef.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description:

This API attaches an explicit chamfer as an attribute of the given face and places support attributes on the left and right support faces, given or deduced. Local operations are then able to make the blend geometry agree with the geometry given in the explicit blend attribute. It gives a means for changing a blend in place (a blend edit) or for making new exact blend geometry in place of existing blend geometry perhaps obtained as the result of earlier modeling operations or imported from elsewhere. The face to be regarded as a blend need not have been made using a blend operation. The change of blend geometry must not be so great as to require a change in the topology of the blend face.

All the arguments except the first are defaulted since in many cases (e.g. where the blend surface is a simple analytic surface) they can be deduced from the given blend face. If arguments are omitted but cannot be deduced from the given blend face, the routine will return a failure outcome.

At present this API routine does not deal with setbacks.

Errors: Left or right range less than –SPAresabs.

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_set_exp_const_round

Function: Blending, Modifying Models

Action: Sets an explicit constant round attribute on a face and support attributes on its support faces.

Prototype:

```
outcome api_set_exp_const_round (
    FACE* bl_face,              // face to be given
                                // attribute (not
                                // defaulted)
    FACE* left_face             // left support face
        = NULL,
    FACE* right_face            // right support face
        = NULL,
    double const& radius        // radius
        =*(double*)NULL_REF,
    logical const& cvxty        // convexity, TRUE
        =*(logical*)NULL_REF,   // for convex
    plane const& mid_plane      // approximate normal
        =*(plane*)NULL_REF,     // plane at mid-point
                                // of spine
    logical const& start_cdt    // start condition
        =*(logical*)NULL_REF,   // TRUE if open
    logical const& end_cdt      // end condition
        =*(logical*)NULL_REF,   // TRUE if open
    AcisOptions* ao             // ACIS
        = NULL                  // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kerngeom/surface/pladef.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API attaches an explicit round as an attribute of the given face and places support attributes on the left and right support faces, given or deduced. Local operations are then able to make the blend geometry agree with the geometry given in the explicit blend attribute. It gives a means of changing a blend in place (a blend edit) or of making new exact blend geometry in place of existing blend geometry perhaps obtained as the result of earlier modeling operations or imported from elsewhere. The face to be regarded as a blend need not have been made using a blend operation. The change of blend geometry must not be so great as to require a change in the topology of the blend face.

All the arguments except the first are defaulted since in many cases (e.g. where the blend surface is a simple analytic surface) they can be deduced from the given blend face. If arguments are omitted but cannot be deduced from the given blend face, the routine returns a failure outcome. At present this API routine does not deal with setbacks.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_set_exp_co_ro_fbl_att

Function: Blending, Modifying Models

Action: Sets an explicit constant round attribute on a face (and any faces split from it) and places support attributes on the support faces.

Prototype:

```
outcome api_set_exp_co_ro_fbl_att (
    ENTITY_LIST const&                 // start cross
        start_cross_coedges,           // coedges
    ENTITY_LIST const&                 // end cross
        end_cross_coedges,             // coedges
    ENTITY_LIST const&                 // spring
        spring_coedges,                // coedges
    logical const&                     // support on
        support_on_left,               // left if TRUE
    double const& bl_radius            // blend radius
        =*(double*)NULL_REF,
    double const& len_tol              // length
        =*(double*)NULL_REF,           // tolerance
    ENTITY_LIST& bl_faces_noted        // blend faces
        =*(ENTITY_LIST*)NULL_REF,
// noted
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "blend/kernapi/api/blendapi.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API attaches an explicit constant round face–face blend attribute to the face of the given coedges (and any other face split from it) and places support attributes on the left and right support faces. If not given, the blend radius is estimated. A def–plane and convexity are found for each attribute made. If blend radius is not given, it is estimated from the blend face curvature.

Member functions of the attributes made are used to re–blend geometry in healing.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_set_exp_co_ro_ffbl_att

Function: Blending, Modifying Models

Action: Sets an explicit constant round attribute on a face (and any faces split from it) and places support attributes on the support faces.

Prototype:

```
outcome api_set_exp_co_ro_ffbl_att (
    ENTITY_LIST const&              // start cross
        start_cross_coedges,        // coedges
    ENTITY_LIST const&              // end cross
        end_cross_coedges,          // coedges
    ENTITY_LIST const&              // left spring
        left_spring_coedges,        // coedges
    ENTITY_LIST const&              // right spring
        right_spring_coedges,       // coedges
    double const& bl_radius         // blend
        =*(double*)NULL_REF,        // radius
    double const& len_tol           // length
        =*(double*)NULL_REF,        // tolerance
    ENTITY_LIST& bl_faces_noted     // blend faces
        =*(ENTITY_LIST*)NULL_REF,
// noted
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API attaches an explicit constant round face–face blend attribute to the face of the given coedges (and any other face split from it) and places support attributes on the left and right support faces. If not given, the blend radius is estimated. A def–plane and convexity are found for each attribute made. If blend radius is not given, it is estimated from the blend face curvature.

Member functions of the attributes made are used to re–blend geometry in healing.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_set_var_blends

Function: Blending, Modifying Models

Action: Sets an implicit variable radius blend on each of a list of edges or vertices.

Prototype:
```
outcome api_set_var_blends (
    ENTITY_LIST const& entities,// entities to be
                                // blended
    double start_radius,        // start blend radius
    double end_radius,          // end blend radius
    double start_setback,       // start setback
    double end_setback,         // end setback
    CURVE* calibration_curve    // calibration curve
        = NULL,
    EDGE* first_edge            // first edge in
        = NULL,                 // smooth sequence
    EDGE* last_edge             // last edge in
        = NULL,                 // smooth sequence
    double                      // setback difference
        start_sbdiff = 0,       // (r – l) at start
    double                      // setback difference
        end_sbdiff = 0,         // (r – l) at end
    logical                     // setback difference
        start_sbdiff_set = TRUE,// set at start
    logical                     // setback difference
        end_sbdiff_set = TRUE,  // set at end
    double start_stop_ang = 0,  // start stop angle
    double end_stop_ang = 0,    // end stop angle
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/geom/curve.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Implicit variable-radius blends are attached as attributes to each of the edges or vertices in the list.

Setback at an end of an edge Determines where the blend is to be stopped short of the vertex at the edge end. It is only significant when the vertex is blended.

The blend radii and left and right setbacks must be supplied in the local space of the body.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_set_vblend

Function: Blending, Modifying Models

Action: Sets an implicit blend on a vertex.

Prototype:

```
outcome api_set_vblend (
    VERTEX* this_vertex,     // vertex to be blended
    double bulge,            // bulge factor [0...2]
    double setback           // setback from vertex
        = 0,
    bl_continuity bl_contin  // blend continuity
        = slope_continuous,
    bl_v_property bl_v_prop  // blend property
        = bl_v_unset,
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”blend/kernbool/blending/vbl_enum.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernbool/blending/bl_enum.hxx”
#include ”kernel/kerndata/top/vertex.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: An implicit blend is attached as an attribute of the given vertex.

The bulge may be given a value between zero and two (one is usual). A larger value increases the fullness of the blend.

The setback, if zero or positive, is applied to the end of each blended edge terminating in the vertex. If the given setback is negative, the setbacks already given to blended edges at the vertex remain unchanged.

The details of the blend face are found from the radius and setbacks of the edges that end in the vertex.

The bl_continuity can be set to: unset_continuity, position_continuous, slope_continuous (default), or curvature_continuous.

The blend property can be set to:

bl_v_cap	means a cap or miter at a vertex joined to one or two blended edges (same as default behavior with no attribute on the vertex at the end of a blend sequence).
bl_v_roll_on	means close off the open end or continue on to the next blended edge at the vertex using edge–face or edge–edge blends.
bl_v_runout	means close off the open end using a variable–radius blend.
bl_v_unset	is a vertex blend.
bl_v_blend	is a vertex blend.

Errors: Pointer to vertex NULL or not to a VERTEX.
Bulge factor less than 0 or greater than 2.0.

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_set_vblend_auto

Function: Blending, Modifying Models

Action: Sets an implicit blend on a vertex, computing and assigning setbacks automatically.

Prototype:

```
outcome api_set_vblend_auto (
    VERTEX* this_vertex,     // vertex to be blended
    double bulge,            // bulge factor [0...2]
    bl_continuity bl_contin // blend continuity
        = slope_continuous,
    bl_v_property bl_v_prop // blend property
        = bl_v_unset,
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”blend/kernbool/blending/vbl_enum.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernbool/blending/bl_enum.hxx”
#include ”kernel/kerndata/top/vertex.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: An implicit blend is attached as an attribute of the given vertex. The bulge may be given a value between zero and two (one is usual). A larger value increases the fullness of the blend.

Setbacks are computed and applied to each edge of the vertex. Setback values depend on the geometry of the body local to the blended vertex and on sizes of blends already assigned to edges ending in the vertex. The caller should ensure that blend attributes are set on the edges of the vertex before this API is called. Setbacks on edges blended with blends of zero size are left unchanged.

The details of the blend face are found from the radius and setbacks of the edges that end in the vertex.

The bl_continuity can be set to: unset_continuity, position_continuous, slope_continuous (default), or curvature_continuous.

The blend property can be set to:

bl_v_cap	means a cap or miter at a vertex joined to one or two blended edges (same as default behavior with no attribute on the vertex at the end of a blend sequence).
bl_v_roll_on	means close off the open end or continue on to the next blended edge at the vertex using edge–face or edge–edge blends.
bl_v_runout	means close off the open end using a variable–radius blend.
bl_v_unset	is a vertex blend.
bl_v_blend	is a vertex blend.

Errors: Pointer to vertex is NULL or not to a VERTEX.
Bulge factor less than 0 or greater than 2.0.

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Changes model

api_set_vblend_autoblend

Function: Blending

Action: Sets an implicit blend on a vertex, computing and assigning setbacks automatically, and using rolling ball vertex blends where possible.

Prototype:

```
outcome api_set_vblend_autoblend (
    VERTEX* this_vertex,    // entities
    double bulge,           // bulge factor [0...2]
    bl_continuity bl_contin // reserved for
        = slope_continuous, // future enhancement
    bl_v_property bl_v_prop // blend property
        = bl_v_unset,
    AcisOptions* ao         // ACIS
        = NULL              // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”blend/kernbool/blending/vbl_enum.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernbool/blending/bl_enum.hxx”
#include ”kernel/kerndata/top/vertex.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function is similar to api_set_vblend_auto, but creates vertex blends that have been marked as autosetback blends using a rolling ball vertex blend whenever possible.

An implicit blend is attached as an attribute of the given vertex. The bulge may be given a value between zero and two (one is usual). A larger value increases the fullness of the blend.

Setbacks are computed and applied to each edge of the vertex. Setback values depend on the geometry of the body local to the blended vertex and on sizes of blends already assigned to edges ending in the vertex. The caller should ensure that blend attributes are set on the edges of the vertex before this API is called. Setbacks on edges blended with blends of zero size are left unchanged.

The details of the blend face are found from the radius and setbacks of the edges that end in the vertex.

The bl_continuity is always set to slope_continuous.

The blend property can be set to:

bl_v_cap	means a cap or miter at a vertex joined to one or two blended edges (same as default behavior with no attribute on the vertex at the end of a blend sequence).
bl_v_roll_on	means close off the open end or continue on to the next blended edge at the vertex using edge–face or edge–edge blends.
bl_v_runout	means close off the open end using a variable–radius blend.
bl_v_unset	is a vertex blend.
bl_v_blend	is a vertex blend.

Errors: Pointer to vertex is NULL or not to a VERTEX.
Bulge factor less than 0 or greater than 2.0.

Limitations:	None
Library:	blend
Filename:	blnd/blend/kernapi/api/blendapi.hxx
Effect:	Changes model

api_smooth_edges_to_curve

Function: Blending, Modifying Models

Action: Generates a calibration curve fit to a list of edges.

Prototype:

```
outcome api_smooth_edges_to_curve (
    ENTITY_LIST const& ents,     // entities
    CURVE*& calibration_curve,   // returned
                                 // calibration curve
    EDGE*& first_edge,           // returned the first
                                 // edge on the curve
    EDGE*& last_edge,            // returned the last
                                 // edge on the curve
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/geom/curve.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API makes a single curve that approximates the given list of smoothly–connected edges. The list need not be sorted upon input. The curve and pointers to the edges corresponding to the start and end of the curve are returned. The curve is returned as a new CURVE with a use count of 1, and must be disposed of properly. first_edge and last_edge are pointers directly into the body. This API is used in blending, for example to find a defining curve for a variable–radius blend sequence, which parameterizes the blend surface and calibrates the radius function. Typical usage would be:

```
CURVE *calibration_curve;
EDGE *first_edge;
EDGE *last_edge;
api_smooth_edges_to_curve(ents, calibration_curve,
    first_edge, last_edge);
```

```
result = api_set_var_blends(ents,
    start_radius, end_radius,
    start_setback, end_setback,
    calibration_curve, first_edge, last_edge,
    0, 0, TRUE, TRUE, start_stop_ang, end_stop_ang);

calibration_CURVE->remove();
```

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernapi/api/blendapi.hxx

Effect: Read-only

api_smooth_edge_seq

Function: Object Relationships, Blending, Modifying Models

Action: Gets a sequence of connected edges for the given edge that meet smoothly; i.e., with tangent continuity.

Prototype:

```
outcome api_smooth_edge_seq (
    EDGE* this_edge,         // an edge in the
                             // sequence
    ENTITY_LIST& edge_list, // returned the list of
                             // connected edges found
    AcisOptions* ao          // ACIS
        = NULL               // options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”blend/kernapi/api/blendapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

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

Limitations: It is assumed that the input edge(s) define natural or trimming edges participating in loops of faces in a solid or sheet model. The code explicitly assumes that the coedges have non-NULL partner and owner pointers. A segmentation error occurs if one passes in edges that only participate in a pure wire body.

Library:	blend
Filename:	blnd/blend/kernapi/api/blendapi.hxx
Effect:	Read-only

api_terminate_blending

Function: Blending, Modeler Control

Action:	Terminates the blending library.
Prototype:	`outcome api_terminate_blending ();`

Includes:	`#include ”kernel/acis.hxx”`
	`#include ”blend/kernapi/api/blendapi.hxx”`
	`#include ”kernel/kernapi/api/api.hxx”`

Description:	Refer to Action.
Errors:	None
Limitations:	None
Library:	blend
Filename:	blnd/blend/kernapi/api/blendapi.hxx
Effect:	System routine

get_ATTRIB_BLEND_TYPE

Function: Blending, Attributes

Action:	Gets the type of blend attribute.
Prototype:	`int get_ATTRIB_BLEND_TYPE ();`

Includes:	`#include ”kernel/acis.hxx”`
	`#include ”blend/geomhusk/blndtype.hxx”`

Description:	Refer to Action.
Errors:	None
Limitations:	None
Library:	blend

Filename: blnd/blend/geomhusk/blndtype.hxx

Effect: Read–only

is_ATTRIB_BLEND

Function: Blending

Action: Determines if an ENTITY is an ATTRIB_BLEND.

Prototype:
```
logical is_ATTRIB_BLEND (
    const ENTITY* e          // entity to test
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”blend/kernbool/blending/bl_att.hxx”
#include ”kernel/kerndata/data/entity.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernbool/blending/bl_att.hxx

Effect: Read–only

is_ATTRIB_BLINFO

Function: Blending

Action: Determines if an ENTITY is an ATTRIB_BLINFO.

Prototype:
```
logical is_ATTRIB_BLINFO (
    const ENTITY* e          // entity to test
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”blend/kernbool/blending/bl_att.hxx”
#include ”kernel/kerndata/data/entity.hxx”
```

Description: Refer to Action.

Errors:	None
Limitations:	None
Library:	blend
Filename:	blnd/blend/kernbool/blending/bl_att.hxx
Effect:	Read–only

is_ATTRIB_CONST_CHAMFER

Function: Blending

Action: Determines if an ENTITY is an ATTRIB_CONST_CHAMFER.

Prototype:

```
logical is_ATTRIB_CONST_CHAMFER (
    const ENTITY* e          // entity to test
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”blend/kernbool/blending/ch_const.hxx”
#include ”kernel/kerndata/data/entity.hxx”
```

Description: Refer to Action.

Errors:	None
Limitations:	None
Library:	blend
Filename:	blnd/blend/kernbool/blending/ch_const.hxx
Effect:	Read–only

is_ATTRIB_CONST_ROUND

Function: Blending

Action: Determines if an ENTITY is an ATTRIB_CONST_ROUND.

Prototype:

```
logical is_ATTRIB_CONST_ROUND (
    const ENTITY* e          // entity to test
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”blend/kernbool/blending/ro_const.hxx”
#include ”kernel/kerndata/data/entity.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernbool/blending/ro_const.hxx

Effect: Read–only

is_ATTRIB_FFBLEND

Function: Blending

Action: Determines if an ENTITY is an ATTRIB_FFBLEND.

Prototype:

```
logical is_ATTRIB_FFBLEND (
    const ENTITY* e          // entity to test
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”blend/kernbool/blending/bl_att.hxx”
#include ”kernel/kerndata/data/entity.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernbool/blending/bl_att.hxx

Effect: Read–only

is_ATTRIB_VAR_BLEND

Function: Blending

Action: Determines if an ENTITY is an ATTRIB_VAR_BLEND.

Prototype:

```
logical is_ATTRIB_VAR_BLEND (
    const ENTITY* e          // entity to test
    );
```

Includes:	`#include ”kernel/acis.hxx”`
	`#include ”baseutil/logical.h”`
	`#include ”blend/sg_husk/vrbln/v_bl_att.hxx”`
	`#include ”kernel/kerndata/data/entity.hxx”`

Description:	Refer to Action.
Errors:	None
Limitations:	None
Library:	blend
Filename:	blnd/blend/sg_husk/vrbln/v_bl_att.hxx
Effect:	Read–only

is_ATTRIB_VBLEND

Function: Blending

Action:	Determines if an ENTITY is an ATTRIB_VBLEND.

Prototype:

```
logical is_ATTRIB_VBLEND (
    const ENTITY* e          // entity to test
    );
```

Includes:	`#include ”kernel/acis.hxx”`
	`#include ”baseutil/logical.h”`
	`#include ”blend/kernbool/blending/vbl_att.hxx”`
	`#include ”kernel/kerndata/data/entity.hxx”`

Description:	Refer to Action.
Errors:	None
Limitations:	None
Library:	blend
Filename:	blnd/blend/kernbool/blending/vbl_att.hxx
Effect:	Read–only

is_ATT_BL_INST

Function: Blending

Action:	Determines if an ENTITY is an ATT_BL_INST.

Prototype:

```
logical is_ATT_BL_INST (
    const ENTITY* e          // entity to test
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "blend/kernbool/entent/bl_inst.hxx"
#include "kernel/kerndata/data/entity.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernbool/entent/bl_inst.hxx

Effect: Read–only

is_BLEND_ANNOTATION

Function: Blending

Action: Determines if an ENTITY is a BLEND_ANNOTATION.

Prototype:
```
logical is_BLEND_ANNOTATION (
    const ENTITY* e          // entity to test
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "blend/kernbool/blend1/blndanno.hxx"
#include "kernel/kerndata/data/entity.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: blend

Filename: blnd/blend/kernbool/blend1/blndanno.hxx

Effect: Read–only

is_BLEND_ANNO_EDGE

Function: Blending

Action: Determines if an ENTITY is a BLEND_ANNO_EDGE.

Prototype:	`logical is_BLEND_ANNO_EDGE (`
	`    const ENTITY* e          // entity to test`
	`    );`

Includes:	`#include ”kernel/acis.hxx”`
	`#include ”blend/kernbool/blend1/blndanno.hxx”`
	`#include ”baseutil/logical.h”`
	`#include ”kernel/kerndata/data/entity.hxx”`

Description:	Refer to Action.
Errors:	None
Limitations:	None
Library:	blend
Filename:	blnd/blend/kernbool/blend1/blndanno.hxx
Effect:	Read–only