Chapter 5. 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_add_guide_curve_si

Function: Skinning and Lofting

Action: Adds a guide curve to a set of skinning profiles.

Prototype:

```
outcome api_add_guide_curve_si (
    AcisSLInterface* obj,    // skinning interface
                                 // object
    EDGE* in_guide,              // guide curve to add
    AcisOptions* opts = NULL     // ACIS options such
                                 // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/skin_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function adds a guide curve to the AcisSkinningInterface.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_add_mapping_curve_sli

Function: Skinning and Lofting

Action: Adds a guide curve to a set of skinning profiles.

Prototype:

```
outcome api_add_mapping_curve_sli (
    AcisSLInterface* obj,    // AcisSLInterface object
    int num,                 // number of position in
                             // mapping curve
    SPAposition* pts,        // list of positions
    AcisOptions* opts = NULL // ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function adds a mapping curve to the AcisSkinningInterface. A mapping curve controls the breakup of the profiles such that the position in the mapping curve are guaranteed to be aligned.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_add_vertex_sli

Function: Skinning and Lofting

Action: Adds a vertex to each wire in a list of wires.

Prototype:

```
outcome api_add_vertex_sli (
    AcisSLInterface* obj,    // AcisSLInterface object
    WIRE* wire,              // wire to add vertex on
    COEDGE* coedge,          // coedge to add vertex
                             // to middle of
    AcisOptions* opts = NULL // ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/coedge.hxx”
#include ”kernel/kerndata/top/wire.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function adds a vertex to the middle of the specified coedge and to each of the corresponding coedges in the wire list.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_align_wires_sli

Function: Skinning and Lofting

Action: Aligns the directions of the wires in the skinning or lofting profiles.

Prototype:

```
outcome api_align_wires_sli (
    AcisSLInterface* obj,    // AcisSLInterface object
    int start = 0,           // Index of starting wire
    int end = 0,             // Index of ending wire
    AcisOptions* opts = NULL // ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description:	Refer to Action.
Errors:	None
Limitations:	None
Library:	skin
Filename:	skin/skin/kernapi/api/skinapi.hxx
Effect:	Changes model

api_breakup_wires_sli

Function: Skinning and Lofting

Action: Creates an equal number of coedges in each wire of the skinning or lofting profiles.

Prototype:

```
outcome api_breakup_wires_sli (
    AcisSLInterface* obj,      // AcisSLInterface object
    int start = 0,             // index of starting wire
    int end = 0,               // index of ending wire
    AcisOptions* opts = NULL// ACIS options such
                               // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description:	Creates an equal number of coedges in each wire of the skinning or lofting profiles. This is required to create the surfaces.
Errors:	None
Limitations:	None
Library:	skin
Filename:	skin/skin/kernapi/api/skinapi.hxx
Effect:	Changes model

api_build_body_sli

Function: Skinning and Lofting

Action: Builds the sheet body from the data in the lofting interface.

Prototype:

```
outcome api_build_body_sli (
    AcisSLInterface* obj,    // AcisSLInterface object
    BODY*& body_out,         // created body
    AcisOptions* opts = NULL// ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_build_edges_sli

Function: Skinning and Lofting

Action: Builds a list of edges that represent the extents of the surfaces if the wires or coedges were to be lofted or skinned.

Prototype:

```
outcome api_build_edges_sli (
    AcisSLInterface* obj,    // AcisSLInterface object
    ENTITY_LIST& edgeList,   // list of created edges
    AcisOptions* opts = NULL// ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function builds a list of edges that represent the extents of the surfaces if the wires were to be skinned at the present configuration. This function is intended to be used as a preview of the potential lofting body that will be created. The application developer may delete this list of edges at any time, allow the end user to move the vertices of a lofting or skinning wire and recreate the edges.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_build_faces_sli

Function: Skinning and Lofting

Action: Builds a list of skinning or lofting faces.

Prototype:

```
outcome api_build_faces_sli (
    AcisSLInterface* obj,    // AcisSLInterface object
    ENTITY_LIST& edgeList,   // returned face list
    AcisOptions* opts = NULL// ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Builds a list of skinning or lofting faces. The returned faces are not stitched together and no sheet body or solid is made.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_clear_guide_curves_sli

Function: Skinning and Lofting

Action: Clears the guide curves in the AcisSkinningInterface.

Prototype:

```
outcome api_clear_guide_curves_sli (
    AcisSkinningInterface* obj, // skinning interface
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/skin_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function clears all the guide curves in the AcisSkinningInterface

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_clear_mapping_curves_sli

Function: Skinning and Lofting

Action: Removes all the mapping curves from the AcisSLInterface.

Prototype:

```
outcome api_clear_mapping_curves_sli (
    AcisSLInterface* obj,   // AcisSLInterface object
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function removes all the mapping curves from the AcisSLInterface.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_collapse_wires_sli

Function: Skinning and Lofting

Action: Deletes a degenerate coedge in each wire of a list of wires.

Prototype:

```
outcome api_collapse_wires_sli (
    AcisSLInterface* obj,    // AcisSLInterface
    ENTITY_LIST& wire_list, // list of modified wires
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function will remove a degenerate coedge in each wire of a list of wires. In order for the coedges to be removed, each wire in the list must contain a degenerate coedge at the same position. That is, if the fifth coedge of wire one is degenerate, it and the other coedges will only be removed if the fifth coedge of every other wire is degenerate as well.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_create_li

Function: Skinning and Lofting

Action: Creates an AcisLoftingInterface object.

Prototype:

```
outcome api_create_li (
    int number_coedges,      // number of coedges
    Loft_Connected_Coedge_List* // coedge
        edgeList,            //  list
    AcisLoftingInterface*& obj,// interface object
    BODY**& wireBodies,      // wire bodies
    skin_options* opts,      // skin options
    AcisOptions* ao          // ACIS options such as
        = NULL               // version and journal
    );
```

```
outcome api_create_li (
    FACE* face1,             // face 1
    double factor1,          // takeoff factor for
                             // coedges on face 1
    FACE* face2,             // face 2
    double factor2,          // takeoff factor for
                             // coedges on face 2
    AcisLoftingInterface*& obj,// interface object
    BODY**& wireBodies,      // wire bodies
    skin_options* opts,      // skin options
    AcisOptions* ao          // ACIS options such as
        = NULL               // version and journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/acis_options.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/loft_intr.hxx”
#include ”skin/sg_husk/skin/skin_opts.hxx”
```

Description: Creates an AcisLoftingInterface object. This object is used as the input to additional skinning and lofting breakup APIs. It contains all the given input and residual data created via the lofting process.

Commonly, the api_create_li takes a data structure called Loft_Connected_Coedge_List. This is composed of the number of coedges in the cross section, a pointer to the first coedge in a list of coedges, the cross section orientation, the take-off vector weight factor, and an array of laws. Every cross section used in lofting has to have this data structure. At the very minimum, two Loft_Connected_Coedge_List structures need to be created.

Loft_Connected_Coedge_Listis defined in sg_husk/skin/skin.hxx:

```
//    struct Loft_Connected_Coedge_List {
//        int n_list;             // number of coedges
//        COEDGE **coedge_list;  // list of coedges
//        REVBIT coedge_orient;  // alignment of coedge
//        double cross_tg_attr;  // take–off vector factor
//        law **law_list;        // list of tangency laws
//    };
```

The default orientation of the Loft_Connected_Coedge_List is the orientation of the first coedge. However, it can be reversed using coedge_orient, the third argument of the data structure. When the coedges in the list are associated with a surface, the cross section orientation uses REVBIT to set the direction “out of” or “into” the starting surface. The expression “out of” means that the lofted surface is going out of the given ending surface, while “into” means that the lofted surface is going into the ending surface. When lofting between two surfaces, the starting surface needs to have its REVBIT set to 0 (for “out of”, the default) and the ending surface needs to have its REVBIT set to 1 (for “into”). When lofting between multiple surfaces, the first surface has its REVBIT set for 0, while all others are set for 1. When lofting between a surface and an edge, REVBIT for both can be 0.

The take-off vector is a tangent vector going out of a given surface and into the lofted surface. The lofted surface is always tangent to the surface bounded by the coedges. The vectors denoting the tangency of the section are scaled to a maximum value of 1.0. The weight factor cross_tg_attr, which is the fourth argument of the Loft_Connected_Coedge_List, is used to scale the magnitude. Its default value is 1.0. A value of zero implies suppression of tangency and negative value results in its reversal.

The weight factor argument in the Loft_Connected_Coedge_List for each cross section is used by the take-off vector. Small values for the weighting of the take-off vector mean that the transition from the tangent to the lofted surface happen abruptly. Large values for the weighting of the take-off vector mean that the transition from the tangent to the lofted surface happen more gradually. Extremely high weight values could result in excessive whipping in the lofted surface, if not a self-intersecting surface.

The ending point of any coedge Loft_Connected_Coedge_List should be the starting point of the next coedge in the list, first and last coedges excepted. In other words, the coedges are ordered in the list as they appear next to each other in model space. There cannot be any gaps between adjacent coedges.

The last argument in Loft_Connected_Coedge_List is an array of law pointers. Each coedge in the coedge list can define a law as a constraint which describes the tangency of the edge. If laws aren't used to control the tangency, the pointer should be NULL.

The api_create_li function accepts the following special options depending on the specific API:

– The *arc_length* option is used to choose arc length or isoparametric parameterization of the skinning surfaces. For isoparametric parameterization, the surface parameter in the V direction follows the cross section curves. For arc length parameterization, the surface parameter follows lines of constant length. The default is isoparametric parameterization. Surfaces resulting from skinning with guide curves support isoparametric parameterization only. The default is FALSE.
– The *no_twist* option may be used to minimize the twist of the surface produced. Twist minimization aligns closed curves such that the start of the second curve is aligned to the start of the first curve. Even if a body's shape is unaffected by twisting, a surface with a twist could produce unexpected results when faceting and rendering. The default is TRUE.
– The *align* option is used to align the direction of the cross section curves such that the normal of the first profile points towards the second profile. All other profiles are aligned to follow the first and second. If the sections are not oriented in the same direction, the alignment option should be used to avoid producing a twisted, self intersecting body. The default is TRUE.
– The *perpendicular* option specifies the direction of the take-off

vector, either perpendicular to the coedge or in the loft direction. The default is FALSE (i.e., in the loft direction).

- The *simplify* option simplifies the surface to a conical surface, if applicable. If all of the cross sections lie on a conical surface (plane, cylinder, cone, sphere, or torus), the conical surface is created instead. The SPAresabs variable is used to determine whether or not the cross section lies on a conical surface. The default is TRUE.
- The *closed* option may be used when the user needs to construct a solid body closed in V. (i.e. a Torus). A solid body will be constructed only when all the wires supplied are closed. At least three profiles must be provided to create a closed body.The default is FALSE.
- The *solid* option may be used when a solid loft must be constructed but a closed body is not desired. When a closed body is not desired, the end wires are capped with planar faces. The default is TRUE.
- The *periodic* option allows to construct loft bodies that are periodic in v. i.e. bodies that close back on themselves smoothly (continuously) at the start and end profiles. This option is activated in the skinning API's by giving the ”closed” option a value of 2. In Scheme, this is achieved by setting the ”periodic” flag to #t. As for the closed option, At least three profiles must be supplied to create a periodic loft body. The default is FALSE.
- The *virtual guide* option my be used in order to have the user defined guides effect the body in a global nature. The default is FALSE.
- When the *merge_wirecoedges* option is set to true, the G1 vertices of the skinning and lofting wire profiles are removed by merging adjacent coedges/edges. This improves operations such as blending and shelling because it reduces the coedge/edge count and the number of surfaces and eliminates near tangent edges. The default is TRUE.
- When the *estimate_loft_tanfacs* option is on, the weight factor for the tangency conditions of the loft will be determined such that it minimizes the average radius of curvature of the lofting surfaces. The resulting bodies should support shelling to greater thickness and also blending of their edges to larger blend radii. The default is FALSE.
- The *match_vertices* option allows to suppress the vertex-matching-algorithm which ensures that all profiles consist of the same number of coedges. A heuristic approach is used to determine which vertex pairs are good matches. Profile coedges are then split where additional vertices are needed. This option is forced to TRUE if the coedge numbers of the profiles are not equal. Its default is TRUE.

The algorithm that minimizes the surface twist (see *no_twist* option) may add vertices to some of the profiles if none of the existing vertices match well enough. The *no_new_twist_vertices* option forces the algorithm to choose matching vertices from the existing vertices. The default is FALSE.

Errors: None

Limitations: The coedge lists are assumed to be simple; i.e. they must be well behaved and non-looping.
The parameterization of the surface in its U direction – the direction of the coedge list – follows the parameterization of the curves underlying the coedges.
Lofting cannot produce a surface in which there is any one point on the surface in which the U and V directions are the same or opposite. That is, in no case can an underlying curve of the user-defined wire body and a lateral edge of the surface generated from the lofting algorithm ever meet tangentially. (Lofting will toggle the perpendicular option for you to avoid this condition.)
End capping (for creation of open solid bodies) is supported only for planar end profiles. If the end capping operation fails then the sheet body constructed is returned.

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_create_si

Function: Skinning and Lofting

Action: Creates an AcisSkinningInterface object.

Prototype:

```
outcome api_create_si (
    ENTITY_LIST&,            // list of wires
    AcisSkinningInterface*&,// interface object
    BODY**& wireBodies,      // array of temporary
                             // wire bodies
    BODY* path = NULL,       // optional skinning path
    skin_options* opts,      // skinning options
    AcisOptions* opts        // ACIS options such as
        = NULL               // version and journal
    );
```

```
outcome api_create_si (
    ENTITY_LIST& wireList,   // list of wires
    AcisSLInterface*& obj,   // interface object
    BODY**& wireBodies,      // array of temporary
                             // wire bodies
    skin_options* opts,      // skin options
    double draft_start,      // start of draft angle
    double draft_end,        // end of draft angle
    double draft_start_mag   // start draft
        = 0.0,               // magnitude
    double draft_end_mag     // end draft
        = 0.0,               // magnitude
    SPAvector start_normal   // start draft
        = SPAvector(0.0,0.0,0.0),// normal
    SPAvector end_normal     // end draft
        = SPAvector(0.0,0.0,0.0),// normal
    AcisOptions*             // ACIS options such as
        = NULL               // version and journal
    );

outcome api_create_si (
    ENTITY_LIST& wireList,   // list of wire bodies
    AcisSLInterface*& obj,   // interface object
    BODY**& wireBodies,      // array of temporary
                             // wire bodies
    skinning_normals _normals,// skinning normals
    skin_options* opts,      // skin options
    AcisOptions* ao          // ACIS options such as
        = NULL               // version and journal
    );

outcome api_create_si (
    ENTITY_LIST& wireList,   // list of wires
    AcisSLInterface*& obj,   // interface object
    BODY**& wireBodies,      // array of temporary
                             // wire bodies
    skinning_ruled ruled,    // ruled skinning
    skin_options* opts,      // skinning options
    AcisOptions* opts        // ACIS options such as
        = NULL               // version and Journal
);
```

```
outcome api_create_si (
    ENTITY_LIST& wireList,   // list of wires
    AcisSLInterface*& obj,   // interface object
    BODY**& wireBodies,      // list of created wire
                             // bodies
    SPAvector* vectors,      // skinning vectors
    int num_vectors,         // number of vectors
    double* magnitudes = NULL,  // list of magnitudes
    int num_magnitudes = 0,  // number of magnitudes
    skin_options* opts = NULL,  // skinning options
    AcisOptions* ao = NULL   // Acis options such as
                             // version and journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/vector.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/skin_intr.hxx”
#include ”skin/sg_husk/skin/skin_opts.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
```

Description: Creates an AcisSkinningInterface object. This object is used as the input to additional skinning and lofting breakup APIs. It contains all the given input and residual data created via the skinning process. There are eight functional variations of skinning each differing by methods of surface control however only seven are supported through this API. Branched skinning is only supported through api_skin_wires.

For a complete overview refer to the Skinning section of Chapter 1, Advanced Surfacing Component. This API has the following variations:

- Basic Skinning
- Skinning with a Path
- Skinning with a Draft Angle
- Skinning to the Planar Normal
- Skinning with Guide Curves (or Virtual Guide Curves set to TRUE)
- Skinning with Ruled Surfaces

In all cases of skinning the cross section profiles are copies; i.e., the originals remain. The profiles can be either all open or all closed. If the user provides a set of closed profiles, the face normals of the skin body point outside, away from the body material. When the user provides a set of open profiles, the face normals of the skin face are oriented along the surface normals, and no attempt is made to change the face normal orientation.

Basic Skinning

Basic skinning accepts wire bodies as cross section profiles over which skin surfaces are defined. Basic skinning allows no control over the resulting surfaces other than the cross section profiles themselves. (The "path" parameter should be NULL for basic skinning.)

The profiles can either be all closed or all open loops. In the case of closed loops the end profiles can be degenerate (a point). For example, basic skinning between a circle and a point will give a spline cone with the "point" being the apex. The cross sections can be in the form of wire bodies or edges, however wire bodies are preferred.

Skinning with a Path

Skinning with a path accepts an additional wire body to control the take off directions of the surface. The path is accepted as a wire body. It does not have to intersect the cross section profiles. Skinning with a path accepts degenerate wires as end points, however it is unlikely that a desirable result will be produced.

Skinning with Guide Curves (Virtual Guides)

Skinning with guide curves accepts a list of guides in addition to the cross section profiles. In the case of this API the guides are accepted via the function: api_add_guide_curve_si. The guide curves "stretch" across the profiles from the first to last and control the surface in the *V* direction.

Skinning To the Planar Normal

Skinning to the planar normal accepts the enumerated type "skinning_normals" as the fourth parameter. This enumerated type has four constants: "first_normal", "last_normal", "ends_normal" and "all_normal". This variation of skinning does not accept degenerate profiles (points) or profiles that do not define a plane (lines). In addition, it does not accept any form of magnitude control (see skinning with a draft angle). An optimum magnitude is calculated for you.

Skinning with Draft Angles

Skinning with draft angles accepts two angles to control the take-off direction of the surface at the first and last profiles. The cross section wire list may contain any number of wires (greater than two) however, the draft angles are only supplied to the first and last profiles. The draft angles are accepted in the form of radians. In addition to the angles themselves all overloaded forms of this variation accept two more doubles to control the magnitude of the draft angles. These values may be zero; in this case we calculate the optimum magnitude for you.

Skinning with draft angles allows the end profiles to be degenerate or single straight lines. In this case, the wire normal used in the calculation of the take-off direction will be computed for you. However, if you wish to supply your own, there are two additional parameters that accept normal vectors for the first and last profiles. These vectors are only accepted in the case where the end profiles are degenerate or do not have their own intrinsic normal. (Lines).

Skinning with a draft does not support the closed option.

Ruled Skinning

Ruled skinning accepts the enumerated type "ruled_skinning". This enumerated type has only one constant: "ruled". This variation accepts degenerate end points. In addition, it supports the closed option. It does not detect self intersections.

Special Options

The api_create_si function accepts the following special options depending on the specific API:

The *arc_length* option is used to choose arc length or isoparametric parameterization of the skinning surfaces. For isoparametric parameterization, the surface parameter in the *V* direction follows the cross section curves. For arc length parameterization, the surface parameter follows lines of constant length. The default is isoparametric parameterization. Surfaces resulting from skinning with guide curves support isoparametric parameterization only. The default is FALSE.

The *no_twist* option may be used to minimize the twist of the surface produced. Twist minimization aligns closed curves such that the start of the second curve is aligned to the start of the first curve. Even if a body's shape is unaffected by twisting, a surface with a twist could produce unexpected results when faceting and rendering. The default is TRUE.

The *align* option is used to align the direction of the cross section curves such that the normal of the first profile points towards the second profile. All other profiles are aligned to follow the first and second. If the sections are not oriented in the same direction, the align option should be used to avoid producing a twisted, self intersecting body. The default is TRUE.

The *simplify* option simplifies the surface to a conical surface, if applicable. If all of the cross sections lie on a conical surface (plane, cylinder, cone, sphere, or torus), the conical surface is created instead. The SPAresabs variable is used to determine whether or not the cross section lies on a conical surface. The default is TRUE.

The *closed* option may be used when the user needs to construct a solid body closed in *V.* (i.e., a torus). A solid body will be constructed only when all the wires supplied are closed. At least three profiles must be provided to create a closed body. The default is FALSE.

The *solid* option may be used when a solid loft must be constructed but a closed body is not desired. When a closed body is not desired, the end wires are capped with planar faces. The default is TRUE.

The *periodic* option allows the construction of loft bodies that are periodic in *V,* i.e., bodies that close back on themselves smoothly (continuously) at the start and end profiles. This option is activated in the skinning APIs by giving the *closed* option a value of 2. In Scheme, this is achieved by setting the periodic flag to TRUE. As for the closed option, at least three profiles must be supplied to create a periodic loft body. The default is FALSE.

The *virtual guide* option may be used in order to have the user defined guides affect the body in a global nature. The default is FALSE.

When the *merge_wirecoedges* option is set to TRUE, the G1 vertices of the skinning and lofting wire profiles are removed by merging adjacent coedges/edges. This improves operations such as blending and shelling because it reduces the coedge/edge count and the number of surfaces, and eliminates near tangent edges. The default is TRUE.

When the *estimate_loft_tanfacs* option is on, the weight factor for the tangency conditions of the loft will be determined such that it minimizes the average radius of curvature of the lofting surfaces. The resulting bodies should support shelling to greater thickness and also blending of their edges to larger blend radii. The default is FALSE.

The *match_vertices* option suppresses the vertex-matching-algorithm which ensures that all profiles consist of the same number of coedges. A heuristic approach is used to determine which vertex pairs are good matches. Profile coedges are then split where additional vertices are needed. This option is forced to TRUE if the coedge numbers of the profiles are not equal. Its default is TRUE.

The algorithm that minimizes the surface twist (see ”no_twist” option) may add vertices to some of the profiles if none of the existing vertices match well enough. The ”no_new_twist_vertices” option allows to force the algorithm to choose matching vertices from the existing vertices. The default is FALSE.

Errors: None

Limitations: The cross section profiles are assumed to be simple; i.e., they must be well behaved and non-looping.

The parameterization of the surface in its U direction – the direction of the user-defined wire bodies – follows the parameterization of the curves underlying the wire bodies.

Skinning cannot produce a surface in which there is any one point on the surface in which the U and V directions are the same or opposite. That is, in no case can an underlying curve of the user-defined wire body and a lateral edge of the surface generated from the skinning algorithm ever meet tangentially.

End capping (for creation of open solid bodies) is supported only for planar end profiles. If the end capping operation fails then the sheet body constructed is returned.

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_delete_sli

Function: Skinning and Lofting

Action: Deletes an AcisSLInterface object.

Prototype:

```
outcome api_delete_sli (
    AcisSLInterface* obj,    // object to delete
    AcisOptions* opts = NULL // ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: The coedge lists are assumed to be simple; i.e. they must be well behaved and non–looping.

The parameterization of the surface in its *u* direction (the direction of the coedge list) follows the parameterization of the curves underlying the coedges.

Skinning cannot produce a surface in which there is any one point on the surface in which the *u* and *v* directions are the same or opposite. That is, in no case can an underlying curve of the user–defined wire body and a lateral edge of the surface generated from the skinning algorithm ever meet tangentially

End capping (for creation of open solid bodies) is supported only for planar end profiles. If the end capping operation fails then the sheet body constructed is returned.

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read–only

api_estimate_min_rad_curvature_skin

Function: Skinning and Lofting

Action: Estimates the magnitude of the tangent vector field to build surfaces with a minimum radius of curvature.

Prototype:

```
outcome api_estimate_min_rad_curvature_skin (
    AcisSLInterface* obj,    // AcisSLInterface object
    double& min_rad,         // minimum radius of
                             // curvature that can be
                             // used
    AcisOptions* opts = NULL // ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API estimates the minimum radius of curvature of the skin surfaces of the body. This should help the user to estimate the maximum thickness that the skin surface can possibly be shelled and the maximum blend radii that can possibly be applied to the edges of the skin body. This API works on both skin and loft bodies.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_estimate_tangent_factor_scale_li

Function: Skinning and Lofting

Action: Estimates the optimal magnitude to scale the takeoff vectors on the loft profile cross section.

Prototype:
```
outcome api_estimate_tangent_factor_scale_li (
    AcisLoftingInterface* obj,  // AcisSLInterface
                                // object
    SPAinterval& range,         // range of
                                // magnitudes for the
                                // takeoff vectors
    double& optimum             // Optimal magnitude
        =*(double*)NULL_REF,    // for the takeoff
                                // vectors
    double& min_radius          // minimum radius of
        =*(double*)NULL_REF,    // curvature of the
                                // loft corresponding
                                // scale factor =
                                // optimum
    AcisOptions* opts = NULL    // ACIS options such
                                // as version/journal
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/vector/interval.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/loft_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API estimates the factor by which all of the takeoff vectors should be scaled to obtain a ”loft” body with the minimum radius of curvature that is possible. It is important to note that this API provides a single scale factor that should be applied simultaneously to all the takeoff vectors. Doing so will, for instance, produce a loft that may be shelled to a greater thickness or allow its edges to be blended with larger blend radii. If the user is only interested in a range of optimum values to apply then the last two arguments to the api should not be supplied. This results in faster execution times. Since the API estimates the magnitudes of the takeoff vectors, it will produce a scale factor equal to 1 (by default) if used with ”skin” bodies.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_get_tangent_factors_li

Function: Skinning and Lofting

Action: Gets the current set of tangent factors on the loft profiles.

Prototype:
```
outcome api_get_tangent_factors_li (
    AcisLoftingInterface* obj,  // AcisSLInterface
                                // object
    double*& tan_factors,       // tangent factors
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/loft_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_initialize_skinning

Function: Modeler Control, Skinning and Lofting

Action: Initializes the skinning library.

Prototype:
```
outcome api_initialize_skinning ();
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: System routine

api_loft_coedges

Function: Skinning and Lofting, Creating Entities

Action: Creates a sheet body that fits a surface through a sequence of coedges, while providing start and end tangent control.

Prototype:

```
outcome api_loft_coedges (
    int n_set,                     // number of
                                   // connected coedge
                                   // lists
    Loft_Connected_Coedge_List*    // list of connected
        set_ce_list,               // coedge lists
    BODY*& sheet_body,             // output sheet body
    skin_options* opts,            // skinning options
    AcisOptions* ao = NULL     // ACIS options such
                               // as version and journal
    );
```

```
outcome api_loft_coedges (
    int n_set,                     // number of
                                   // connected coedge
                                   // lists
    Loft_Connected_Coedge_List*    // list of connected
        set_ce_list,               // coedge lists
    int num_of_guides              // number of guides
    EDGE*guides[]                  // guide curves
    BODY*& sheet_body,             // output sheet body
    skin_options* opts,            // skinning options
    AcisOptions* ao = NULL     // ACIS options such
                               // as version and journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/skin_opts.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: The newly created shell or solid body interpolates *n* cross sections defined by a list of coedges. The *n* sections are given as an array of cross section definitions (Loft_Connected_Coedge_List). Each array element defines a list of coedges whose edges define the cross section. If the coedges exist in a FACE, the surface tangents at the boundary are associated with the sections. The default direction of tangents is outwards from the face.

The api_loft_coedges takes a data structure called Loft_Connected_Coedge_List. This is composed of the number of coedges in the cross section, a pointer to the first coedge in a list of coedges, the cross section orientation, the take-off vector weight factor, and an array of laws. Every cross section used in lofting has to have this data structure. At the very minimum, two Loft_Connected_Coedge_List structures need to be created.

In the case that a closed solid body is required, at least three distinct coedge lists must be provided. If the first and last coedge lists are identical, a closed solid loft body is automatically constructed. The end tangency will depend on the laws specified at the first and last profiles. If duplicate copies of the internal coedge lists are sent to api_loft_coedges, the copies are ignored while producing the loft.

If the user provides a set of closed profiles, the face normals of the loft body point outside, away from the body material. When the user provides a set of open profiles, the face normals of the loft face are oriented along the surface normals, and no attempt is made to change the face normal orientation.

Loft_Connected_Coedge_List is defined in sg_husk/skin/skin.hxx:

```
struct Loft_Connected_Coedge_List {
    int n_list;            // number of coedges
    COEDGE **coedge_list;  // list of coedges
    REVBIT coedge_orient;  // alignment of coedge
    double cross_tg_attr;  // take-off vector factor
    law **law_list;        // list of tangency laws
};
```

The default orientation of the Loft_Connected_Coedge_List is the orientation of the first coedge. However, it can be reversed using coedge_orient, the third argument of the data structure. When the coedges in the list are associated with a surface, the cross section orientation uses REVBIT to set the direction “out of” or “into” the starting surface. The expression “out of” means that the lofted surface is going *out of* the given ending surface, while “into” means that the lofted surface is going *into* the ending surface. When lofting between two surfaces, the starting surface needs to have its REVBIT set to 0 (for “out of”, the default) and the ending surface needs to have its REVBIT set to 1 (for “into”). When lofting between multiple surfaces, the first surface has its REVBIT set for 0, while all others are set for 1. When lofting between a surface and an edge, REVBIT for both can be 0.

The take-off vector is a tangent vector going out of a given surface and into the lofted surface. The lofted surface is always tangent to the surface bounded by the coedges. The vectors denoting the tangency of the section are scaled to a maximum value of 1.0. The weight factor cross_tg_attr, which is the fourth argument of the Loft_Connected_Coedge_List, is used to scale the magnitude. Its default value is 1.0. A value of zero implies suppression of tangency and negative value results in its reversal.

The weight factor argument in the Loft_Connected_Coedge_List for each cross section is used by the take-off vector. Small values for the weighting of the take-off vector mean that the transition from the tangent to the lofted surface happen abruptly. Large values for the weighting of the take-off vector mean that the transition from the tangent to the lofted surface happen more gradually. Extremely high weight values could result in excessive whipping in the lofted surface, if not a self-intersecting surface.

The ending point of any coedge Loft_Connected_Coedge_List should be the starting point of the next coedge in the list, first and last coedges excepted. In other words, the coedges are ordered in the list as they appear next to each other in model space. There cannot be any gaps between adjacent coedges.

The last argument in Loft_Connected_Coedge_List is an array of law pointers. Each coedge in the coedge list can define a law as a constraint which describes the tangency of the edge. If laws aren’t used to control the tangency, the pointer should be NULL.

Once the cross section data structures have been established, the api_loft_coedges function can be called. However, if needed, guide curves can be passed to the api. Lofting with guides displays the following behavior.

Lofting with guide curves accepts an array of guides in addition to the cross section profiles. The guides are accepted as EDGEs only, never as wire bodies. The curve underlying the guide must be c1 continuous and non–looping. In addition, the guide must touch each wire profile within SPAresabs and start and stop exactly on the first and last profiles. Direction is not important. The guides can intersect the section profiles on or off vertices. If the guide does not lie on any vertices no surface edge is created. If the guide intersects the first profile off a vertex but intersects any other profile on a vertex, a surface edge is made. In addition to accepting guides the overloaded API supporting guide curves takes the logical flag ”VirtualGuides”. This flag toggles the guides from being local surface control to global surface control. For example, in the case of local surface control, the guide curve only affects the surface created from the edges of the wire that the guide intersects. In the case of ”VirtualGuides” being TRUE, the guide affects all the surfaces made from the wire body cross sections.

Lofting with guide curves does not support degenerate wires because tangent constraints cannot be placed on degenerate profiles. To use guides on degenerate profiles, please see api_skin_wires.

The api_loft_coedges function accepts eleven special options:

– arc_length_option can be used to choose either arc length or isoparametric parameterization. If the series of cross sections are equispaced, the isoparametric option may be used. If the sections do not have the same number of coedges, the arc length option produces a better result. With arc length parameterization, the proportional lengths of each coedge on each section are mapped to the other section. These markings then serve as boundaries for additional coedges that are created on each section. The default is FALSE.
– twist_option may be used to minimize the twist of the surface produced. The default is FALSE.
– align_direction_option can align the direction of the cross section curves such that they are in the same direction as the first curve. If the sections are not oriented in the same direction, the alignment option should be used to avoid producing a twisted, self intersecting body. The default is TRUE.
– perpendicular_option specifies the direction of the take-off vector, either perpendicular to the coedge or in the skin direction. The default is FALSE (i.e., in the skin direction), because a perpendicular take-off vector can cause self-intersections to the surface.
– simplify_option simplifies the surface to a conical surface, if applicable. If all of the cross sections lie on a conical surface (plane,

cylinder, cone, sphere, or torus), the conical surface is created instead. The SPAresabs variable is used to determine whether or not the cross section lies on the conical surface. The default is FALSE.

– closed_option (closed = 1/TRUE) may be used when the user needs to construct a body closed in v. A solid body will be constructed only when all the wires supplied are closed; otherwise this option will be ignored. If the user provides a value closed = 2, the loft body will be periodic in v, instead of being closed in v. The default is FALSE i.e. closed = 0.
– solid_option may be used when a solid loft must be constructed but a closed body is not desired. When a closed body is not desired, the end wires are capped with planar faces. The default is FALSE.
– virtual guide option my be used in order to have the user defined guides effect the body in a global nature. The default is FALSE.
– When the merge_wirecoedges option is set to true, the G1 vertices of the skinning and lofting wire profiles are removed by merging adjacent coedges/edges. This improves operations such as blending and shelling because it reduces the coedge/edge count and the number of surfaces and eliminates near tangent edges. The default is TRUE.
– match_vertices option suppresses the vertex–matching–algorithm which ensures that all profiles consist of the same number of coedges. A heuristic approach is used to determine which vertex pairs are good matches. Profile coedges are then split where additional vertices are needed. This option is forced to TRUE if the coedge numbers of the profiles are not equal. Its default is TRUE.
– guidePreference option specifies the surface constraint that will be used if the guides and tangent vectors do not match. The two possible values are constrain_to_guide and constrain_to_tangent. The default is constrain_to_guide.

Errors: None

Limitations: End capping (for creation of open solid bodies) is supported only for planar end profiles. If the end capping operation fails then the sheet body constructed is returned. When end capping for constructing solid lofts, it is possible to produce self intersecting faces and therefore invalid bodies. Self intersection checks are not being done with this function.

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_loft_faces

Function: Skinning and Lofting, Creating Entities

Action: Unites two bodies using lofting between two faces.

Prototype:

```
outcome api_loft_faces (
    FACE* face1,                  // first face
    double fact1,                 // take off factor
                                  // for coedges on
                                  // face1
    FACE* face2,                  // second face
    double fact2,                 // take off factor
                                  // for coedges on
                                  // face1
    BODY*& body,                  // solid body
                                  // returned
    skin_options* opts,           // skinning options
    AcisOptions* ao = NULL    // ACIS options such
                              // as version/journal
    );

outcome api_loft_faces (
    FACE* face1,                  // first face
    double fact1,                 // take off factor
                                  // for coedges on
                                  // face1
    FACE* face2,                  // second face
    double fact2,                 // take off factor
                                  // for coedges on
                                  // face2
    int num_of_guides,            // number of guides
    EDGE* guides[],               // guides
    BODY*& body,                  // solid body
                                  // returned
    skin_options* opts,           // skinning options
    AcisOptions* ao = NULL        // ACIS options such
                                  // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/skin_opts.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: When a solid is lofted, the face provided as an argument is removed from the body. (Which is one of the differences between lofting between solids and sheets.) Peripheral edges from the first face are used to skin to the peripheral edges of the second face. The result is one body. However, this function does not conduct a self–intersecting check on that body.

Lofting uses the surface information from the adjacent surfaces to determine take-off vectors. (Take–off vectors are pre–established and cannot be changed.) Skinning does not use take-off vectors.

When ACIS makes a BODY from a FACE using api_loft_faces, it makes a single-sided face body. A single-sided face body is really a solid, not a sheet (infinitely thin) body. When a face body is single-sided, basic solid modeling concepts (and ACIS) consider the body to be a solid which extends from the back side of the face out to infinity with ill–defined boundaries extending where the edges of the original face extend backward. Specific use of the single–sided face body determines whether subsequent operations, such as Booleans, work properly.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_lose_surface_conditions_li

Function: Skinning and Lofting

Action: Removes the surface conditions from the wires in the lofting profiles.

Prototype:

```
outcome api_lose_surface_conditions_li (
    AcisLoftingInterface* obj,  // AcisSLInterface
                                // object
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/loft_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Removes the surface conditions from the wires in the lofting profiles. This will effectively reduce the loft to a skinning operation.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_make_mapping_curves_sli

Function: Skinning and Lofting

Action: Gets a list of the mapping curves that currently exist in the AcisSLInterface.

Prototype:
```
outcome api_make_mapping_curves_sli (
    AcisSLInterface* obj,    // AcisSLInterface object
    ENTITY_LIST& edgeList,   // List of mapping curves
                             // that are edges
    AcisOptions* opts = NULL // ACIS options such
                             // as version/journal
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function returns a list of edges that represent the mapping curves in the AcisSLInterface.

Errors: The mapping curve is bad; possibly intersecting another mapping curve.

Limitations: None

Library:	skin
Filename:	skin/skin/kernapi/api/skinapi.hxx
Effect:	Read-only

api_make_wires_sli

Function: Skinning and Lofting

Action: Creates a set of broken up wires used for skinning or lofting.

Prototype:

```
outcome api_make_wires_sli (
    AcisSLInterface* obj,    // AcisSLInterface object
    ENTITY_LIST& wire_list, // list of broken–up
                            // wires
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Creates a set of wires that are broken–up according to the skinning and lofting breakup algorithm. (The original wires or coedges that were sent in for input still exist.) It is this list of broken up wires that will be used in the skinning or lofting algorithm.

Errors:	None
Limitations:	None
Library:	skin
Filename:	skin/skin/kernapi/api/skinapi.hxx
Effect:	Changes model

api_minimize_twist_wires_sli

Function: Skinning and Lofting

Action: Aligns the start vertices of the wires in the skinning/lofting profiles.

Prototype:

```
outcome api_minimize_twist_wires_sli (
    AcisSLInterface* obj,   // AcisSLInterface object
    int start               // index of starting
        = 0,                // profile to be aligned
    int end                 // index of ending
        = 0,                // profile to be aligned
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Performs an alignment of the start vertices of the wires in the skinning/lofting profiles. This allows a surface of minimum twist to be built.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_modify_wire_sli

Function: Skinning and Lofting

Action: Modifies the position of a vertex on a coedge of a wire.

Prototype:

```
outcome api_modify_wire_sli (
    AcisSLInterface* obj,   // AcisSLInterface object
    COEDGE* coedge1,        // coedge to left
                            // of position
    COEDGE* coedge2,        // coedge to right
                            // of position
    SPAposition adjusted_point, // position to adjust
                            // vertex to
    WIRE* wire,             // wire this belongs to
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/coedge.hxx”
#include ”kernel/kerndata/top/wire.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function modifies the position of a vertex on a coedge of a wire. It is used to alter the coedges of a wire of the skinning process so the end–user may control the body generated.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_move_vertex_sli

Function: Skinning and Lofting

Action: Modifies the position of a vertex on an intermediate skinning or lofting wire.

Prototype:

```
outcome api_move_vertex_sli (
    AcisSLInterface* obj,    // AcisSLInterface object
    COEDGE* coedge1,         // first coedge
    COEDGE* coedge2,         // second coedge
    SPAposition pos,         // position of vertex
                             // to be moved
    WIRE* wire,              // Specific skinning or
                             // lofting intermediate
                             // wire containing the
                             // coedges
    AcisOptions* opts = NULL// ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/coedge.hxx”
#include ”kernel/kerndata/top/wire.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API modifies the position of a vertex on an intermediate skinning or lofting wire. It can only be used in this context. Additionally, only non-corner vertices can be moved.

Errors: A non–movable vertex is specified.

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_net_sections

Function: Skinning and Lofting

Action: Creates a sheet body that interpolates a series of sections.

Prototype:

```
outcome api_net_sections (
    int nv_sect,             // number of sections
    Loft_Connected_Coedge_List** // array
        v_sects,             // of coedges
                             // with laws
    int nu_sect,             // number of sections
    Loft_Connected_Coedge_List** // array
        u_sects,             // of coedges
                             // with laws
    BODY*& out_body,         // sheet body returned
    logical simplify         // simplify
        = FALSE,             // geometry toggle
    double net_tol           // net surface intersect.
        = SPAresfit,         // tolerance
    AcisOptions* ao = NULL   // ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This class interpolates a surface through a network of bi-directional curves. The given sections define the cross–sections to be interpolated by the resulting sheet body. There must be at least four sections or bodies, two in each direction. The start points of the curves in the v direction must lie on the first curve in the u direction, and vice versa. The end points of the v curves must lie in the last curve in the u direction, and vice versa. All curves must intersect each other within SPAresabs in the interior of the surface. The sections/bodies are assumed to be simple, meaning only the first section of each body is used for the net surface. The sections can be open or closed. The sections are copies (the originals remain). The sections should be C1 continuous tangent.

If all of the curves (sections) intersect, then the surface passes through the curves and their intersections. If any of the u curves of the net do not intersect all of v curves at some point, the intersection is interpolated. The tolerance value for function is SPAresfit. If the tolerance is changed, a net surface can be created as long as the distance between u curves or between v curves (e.g., in the skin direction) is larger than the tolerance intersection distance between a u curve and a v curve. Thus, the curve interpolating accuracy of the net surface is controlled by the user’s accuracy of the intersections of the cross-sections.

The optional argument align-directions specifies whether or not to line up the directions of the curves in both the u and v directions listed in the body–list. The default is FALSE, which means that the sections are left in the directions that the user originally specified. When this option is TRUE, the directions of open and closed curves in the u or v direction are changed to line up with the first curves in the given direction. After all of the curves in the u and v directions are aligned, the order and directions of the v curves are modified to insure the proper surface parameter orientation. This insures that the start of the first v curve intersects the start of the first u curve. The start of the curve in the v direction is changed to be the start of the curve in the u direction.

The optional argument simplify specifies whether or not to simplify the resulting surface geometry. The default is FALSE, which means not to simplify. If this option is TRUE and if all of the the body-list elements are in the same plane, the result is a planar-surface bounded by the first and last uv sections. If this option is FALSE, the result is a planar net surface.

Errors:	None
Limitations:	None
Library:	skin
Filename:	skin/skin/kernapi/api/skinapi.hxx
Effect:	Changes model

api_net_wires

Function: Skinning and Lofting, Creating Entities

Action: Creates a sheet body that fits a surface through a mesh of wires contained in an array of bodies.

Prototype:

```
outcome api_net_wires (
    int num_v_wires,            // number of wires in
                                // array
    BODY* v_wires[],            // array of wires
    int num_u_wires,            // number of wires in
                                // array
    BODY* u_wires[],            // array of wires
    BODY*& sheet_body,          // sheet body
                                // returned
    skin_options* opts,         // skinning options
    double net_tol              // net surface
        = SPAresfit,            // intersection
                                // tolerance
    AcisOptions* ao = NULL  // ACIS options such
                            // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/skin_opts.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: The given wires define the cross sections to be interpolated by the resulting sheet body. There must be at least two wires.

The wires are copies; i.e., the originals remain.

The wires can be either all open or all closed.

If only two wires are given, one wire may be a degenerate line; i.e., treated as a point. For example, skinning between a degenerate line and a circular arc will give a spline cone with the “point” being the apex.

If all of the curves intersect, then the surface passes through the curves and their intersections. If any of the *u* curves of the net do not intersect all of *v* curves at some point, the intersection is interpolated. The maximum distance for the interpolation is governed by the net_tol argument. The default for this tolerance value is SPAresfit. If the tolerance is changed, a net surface can be created as long as the distance between *u* curves or between *v* curves (e.g., in the skin direction) is larger than the tolerance intersection distance between a *u* curve and a *v* curve. Thus, the curve interpolating accuracy of the net surface is controlled by the user’s accuracy of the intersections of the cross sections.

Errors: A wire pointer in the array is NULL or not to a wire body.

Limitations: The wire bodies are assumed to be simple; i.e., only the first wire of each body is used for skinning. Each wire must have continuous tangents (C1). Wires cannot share an end point (end points must be distinct). The surface and lateral curves are always splines; i.e., the function is not smart about using analytic representations in special cases. A prop edge is inserted when the wires are closed.

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_reenter_coedges_li

Function: Skinning and Lofting

Action: Sets the coedge list and remakes the lofting wires.

Prototype:

```
outcome api_reenter_coedges_li (
    int number_coedges,          // number of coedges
    Loft_Connected_Coedge_List*  // list of new
        edgeList,                // coedges to use
    AcisLoftingInterface* obj,   // AcisSLInterface
                                 // object
    BODY**& wireBodies,          // wire bodies made
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/loft_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Deletes the laws, sections, fixed wire list, internal coedge list, then reassigns the coedges and remakes the wires.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx
skin/skin/sg_husk/api/loft.cxx

Effect: Changes model

api_remove_mapping_curve_sli

Function: Skinning and Lofting

Action: Removes a mapping curve from the AcisSLInterface.

Prototype:

```
outcome api_remove_mapping_curve_sli (
    AcisSLInterface* obj,   // AcisSLInterface object
    int index,              // index of mapping curve
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function removes a mapping curve from the AcisSLInterface. The mapping curve removed is given by the second parameter index.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_remove_vertex_sli

Function: Skinning and Lofting

Action: Removes a vertex from each wire in a list of wires.

Prototype:

```
outcome api_remove_vertex_sli (
    AcisSLInterface* obj,    // AcisSLInterface object
    WIRE* wire,              // wire to remove vertex
                             // from
    SPAposition pos,         // vertex to be removed
    AcisOptions* opts = NULL // ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/wire.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function removes the specified vertex from the its coedge and a vertex from each of the corresponding coedges in the wire list.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_set_options_li

Function: Skinning and Lofting

Action: Sets the options in the lofting interface object.

Prototype:

```
outcome api_set_options_li (
    AcisLoftingInterface* obj,  // interface object
    skin_options* opts,         // skinning options
    AcisOptions* ao = NULL   // ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/loft_intr.hxx”
#include ”skin/sg_husk/skin/skin_opts.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read–only

api_set_options_si

Function: Skinning and Lofting

Action: Sets the options in the skinning interface object.

Prototype:

```
outcome api_set_options_si (
    AcisSkinningInterface* obj, // interface object
    skin_options* opts,         // skinning options
    AcisOptions* ao = NULL   // ACIS options such
                             // as version/journal
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/skin_intr.hxx”
#include ”skin/sg_husk/skin/skin_opts.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read–only

api_set_tangent_factors_li

Function: Skinning and Lofting

Action: Sets the scale factors of the takeoff vectors for the lofting operation.

Prototype:
```
outcome api_set_tangent_factors_li (
    AcisLoftingInterface* obj,  // AcisSLInterface
                                // object
    double* tan_factors,        // tangent factors
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/loft_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API resets the scale factors of the takeoff vectors for the lofting operation. It allows the user to determine these values optimally and reset the values in the skinning/lofting interface object.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_show_guides_si

Function: Skinning and Lofting

Action: Gets a list of the virtual guide curves that currently exist in the AcisSkinningInterface.

Prototype:

```
outcome api_show_guides_si (
    AcisSkinningInterface* obj, // AcisSLInterface
                                // object
AcisSkinningInterface object    // list of
    ENTITY_LIST& edgeList,      // virtual guides
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/skin_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function returns a list of edges that represent the virtual guides in the AcisSkinningInterface.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read-only

api_simplify_wires_sli

Function: Skinning and Lofting

Action: Reduces G1 vertices in a wire body.

Prototype:
```
outcome api_simplify_wires_sli (
    AcisSLInterface* obj,    // interface object
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This function removes the G1 vertices of the intermediate wire profiles by merging the adjacent coedges and edges. This reduces the coedge/edge count, which reduces the number of surfaces as well as eliminating near tangent edges, which improves blending and shelling.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_skin_wires

Function: Skinning and Lofting, Creating Entities

Action: Creates a sheet body that fits a surface (or set of surfaces) through a sequence of wires contained in an array of bodies.

Prototype:
```
outcome api_skin_wires (
    int num_wires,             // no. of wire bodies
    BODY* wires[],             // array of wire bodies
    BODY*& sheet_body,         // resulting sheet body
                               // or solid
    skin_options* opts,        // skinning options
    AcisOptions* ao            // ACIS options such as
        = NULL                 // version / Journal
    );
```

```
outcome api_skin_wires (
    int num_entities,        // no. of wire bodies
    ENTITY_LIST& in_entities,// list of wire bodies
    BODY*& sheet_body,       // resulting sheet body
    skin_options* opts,      // skin options
    AcisOptions* ao          // ACIS options such as
        = NULL               // version and journal
    );

outcome api_skin_wires (
    int num_wires,           // number of wires
    BODY* wires[],           // array of wire bodies
    int no_of_guides,        // number of guide curves
    EDGE*guides[],           // array of edges
    BODY*& sheet_body,       // resulting sheet
                             // body or solid
    skin_options* opts,      // skinning options
    AcisOptions* ao          // ACIS options such as
        = NULL               // version / Journal
    );

outcome api_skin_wires (
    ENTITY_LIST& wire_list,  // list of wire bodies
    ENTITY_LIST& guide_list, // list of guide curves
    BODY*& sheet_body,       // resulting sheet
                             // body or solid
    skin_options* opts,      // skin options
    AcisOptions* ao          // ACIS options such as
        = NULL               // version and journal
    );
```

```
outcome api_skin_wires (
    int num_wires,             // number of wire bodies
    BODY* wires[],             // list of wire bodies
    double draft_start,        // draft angle of first
                               // wire
    double draft_end,          // draft angle of last
                               // wire
    double draft_start_mag,    // magnitude of first
                               // draft
    double draft_end_mag,      // magnitude of second
                               // draft
    BODY*& sheet_body,         // resulting sheet
                               // body or solid
    skin_options* opts,        // skinning options
    AcisOptions* ao            // ACIS options such as
        = NULL                 // version and Journal
    );
```

```
outcome api_skin_wires (
    int num_wires,             // number of wire bodies
    BODY* wires[],             // list of wire bodies
    double draft_start,        // draft angle of first
                               // wire
    double draft_end,          // draft angle of last
                               // wire
    double draft_start_mag,    // magnitude of first
                               // draft
    double draft_end_mag,      // magnitude of second
                               // draft
    SPAvector start_normal,    // normal plane for first
                               // wire
    SPAvector end_normal,      // normal plane for last
                               // wire
    BODY*& sheet_body,         // resulting sheet body
                               // or solid
    skin_options* opts,        // skinning options
    AcisOptions* ao            // ACIS options such as
        = NULL                 // version and Journal
    );
```

```
outcome api_skin_wires (
    int num_entities,       // no. of wire bodies
    ENTITY_LIST& in_entities,// list of wire bodies
    BODY* path,             // path option
    BODY*& sheet_body,      // resulting sheet body
    skin_options* opts,     // skin options
    AcisOptions* ao         // ACIS options such as
        = NULL              // version and journal
    );

outcome api_skin_wires (
    int num_wires,          // no. of wire bodies
    BODY* wires[],          // array of wire bodies
    BODY* path,             // path option
    BODY*& sheet_body,      // resulting sheet body
    skin_options* opts,     // skin options
    AcisOptions* ao         // ACIS options such as
        = NULL              // version and journal
    );

outcome api_skin_wires (
    int num_wires,          // no. of wire bodies
    BODY* wires[],          // array of wire bodies
    BODY*& sheet_body,      // resulting sheet bodies
    skinning_ruled ruled,   // ruled option
    skin_options* opts,     // skin options
    AcisOptions* ao         // ACIS options such as
        = NULL              // version and journal
    );

outcome api_skin_wires (
    int num_wires,          // number of wire bodies
    BODY** wires,           // array of wire bodies
    int num_branches        ,   // number of branches
    int* count_list,        // list of number of wire
                            // bodies per branch
    BODY*** branches,       // array of array of wire
                            // bodies
    BODY*& sheet_body,      // resulting sheet body
                            // or solid
    skinning_normals in_normals,// skinning normals
    skin_options* opts,     // skinning options
    AcisOptions* ao         // ACIS options such
        = NULL              // as version or journal
    );
```

```
outcome api_skin_wires (
    int num_wires,             // number of wire bodies
    BODY** wires,              // array of wire bodies
    BODY*& sheet_body,         // resulting sheet body
                               // or solid body
    skinning_normals in_normals,// skinning normals
    skin_options* opts,        // skinning options
    AcisOptions* ao            // ACIS options such
        = NULL                 // as version and journal
    );

outcome api_skin_wires (
    int num_wires,             // number of wire bodies
    BODY** wires,              // list of wire bodies
    BODY*& sheet_body,         // resulting sheet body
                               // or solid body
    SPAvector* vectors,        // skinning vectors
    int num_vectors,           // number of vectors
    EDGE** guides,             // array of guides
    int number_of_guides,      // number of guides
    skin_options* opts = NULL,  // skinning options
    AcisOptions* ao            // ACIS options such
        = NULL                 // as version and journal
    );

outcome api_skin_wires (
    int num_wires,             // number of wire bodies
    BODY** wires,              // list of wire bodies
    BODY*& sheet_body,         // resulting sheet body
                               // or solid body
    SPAvector* vectors,        // skinning vectors
    int num_vectors,           // number of vectors
    double* magnitudes = NULL,  // magnitudes
    int num_magnitudes = 0,    // number of magnitudes
    skin_options* opts = NULL,  // skinning options
    AcisOptions* ao            // ACIS options such
        = NULL                 // as version and journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/vector.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/skin_opts.hxx”
```

Description: The given wires define the cross sections to be interpolated by the resulting sheet body. The resulting body will contain one or more skinned surfaces. (With the exception of the simplification flag being TRUE, in which the surface(s) might be converted to an applicable conical type.) The skinned surface is a two-parameter surface (*U*,*V*) in which the user defined cross sections provide the *U* parameter of the surface and the skin surface algorithm defines the *V* parameter.

The functional variations of skinning differ by methods of surface control. For a complete overview refer to the Skinning section of Chapter 1, Advanced Surfacing Component. This API has the following variations:

- Basic Skinning
- Skinning with a Path
- Skinning with a Draft Angle
- Skinning with a Draft Angle and Guide Curves
- Skinning to the Planar Normal
- Skinning with Vectors
- Skinning with Vectors and Guide Curves
- Skinning with Guide Curves (or Virtual Guide Curves set to TRUE)
- Skinning with Ruled Surfaces
- Branched Skinning

In all cases of skinning the cross section profiles are copies; i.e., the originals remain. The profiles can be either all open or all closed. If the user provides a set of closed profiles, the face normals of the skin body point outside, away from the body material. When the user provides a set of open profiles, the face normals of the skin face are oriented along the surface normals, and no attempt is made to change the face normal orientation.

Basic Skinning

Basic skinning accepts wire bodies as cross section profiles over which skin surfaces are defined. Basic skinning allows no control over the resulting surfaces other than the cross section profiles themselves. (The "path" parameter should be NULL for basic skinning.)

The profiles can either be all closed or all open loops. In the case of closed loops the end profiles can be degenerate (a point). For example, basic skinning between a circle and a point will give a spline cone with the "point" being the apex. The cross sections can be in the form of wire bodies or edges, however wire bodies are preferred.

Skinning with Guide Curves (Virtual Guides)

Skinning with guide curves accepts a list of guides in addition to the cross section profiles. In the case of this API the guides are accepted via the function: api_add_guide_curve_si. The guide curves "stretch" across the profiles from the first to last and control the surface in the *V* direction.

The guides are accepted as EDGEs never as wire bodies. The curve underlying the guide must be c1 continuous and non-looping. In addition, the guide must touch each wire profile within SPAresabs and start and stop exactly on the first and last profiles. Direction is not important. The guides can intersect the section profiles on or off vertices. If the guide does not lie on any vertices no surface edge is created. If the guide intersects the first profile off a vertex but intersects any other profile on a vertex, a surface edge is made. In addition to accepting guides the overloaded API supporting guide curves takes the logical flag "VirtualGuides". This flag toggles the guides from being local surface control to global surface control. For example, in the case of local surface control, the guide curve only affects the surface created from the edges of the wire that the guide intersects. In the case of "VirtualGuides" being TRUE, the guide affects all the surfaces made from the wire body cross sections.

Skinning with guide curves supports degenerate wires for the first and last wire only.

Skinning with Draft Angles

Skinning with draft angles accepts two angles to control the take-off direction of the surface at the first and last profiles. The cross section wire list may contain any number of wires (greater than two) however, the draft angles are only supplied to the first and last profiles. The draft angles are accepted in the form of radians. In addition to the angles themselves all overloaded forms of this variation accept two more doubles to control the magnitude of the draft angles. These values may be zero; in this case we calculate the optimum magnitude for you.

Skinning with draft angles allows the end profiles to be degenerate or single straight lines. In this case, the wire normal used in the calculation of the take-off direction will be computed for you. However, if you wish to supply your own, there are two additional parameters that accept normal vectors for the first and last profiles. These vectors are only accepted in the case where the end profiles are degenerate or do not have their own intrinsic normal. (Lines).

Skinning with a draft does not support the closed option.

Skinning With Path

Skinning with a path accepts an additional wire body to control the take off directions of the surface. The path is accepted as a wire body. It does not have to intersect the cross section profiles. Skinning with a path accepts degenerate wires as end points, however it is unlikely that a desirable result will be produced.

Skinning To the Planar Normal

Skinning to the planar normal accepts the enumerated type "skinning_normals" as the fourth parameter. This enumerated type has four constants: "first_normal", "last_normal", "ends_normal" and "all_normal". At the specified profiles (i.e., the first or the last or the first and the last or all) the resulting skin surface will take off with the direction that is given by the normal of that profile. This variation of skinning does not accept degenerate profiles (points) or profiles that do not define a plane (lines). In addition, it does not accept any form of magnitude control (see skinning with a draft angle). An optimum magnitude is calculated for you.

Skinning with Vectors

Skinning with vectors accepts as additional constraints a list of vectors and optionally a list of magnitudes to control the take–off directions with which the skin surface leaves the profiles. The number of supplied vectors has to equal the number of profiles if the closed option is set to ”FALSE”. If the closed option is set to ”TRUE”, one more vector can be given. It will be used on the copy that is made of the first profile to create a closed body. If no additional vector is provided the first vector will be used also on the first profile's copy. In a similar way, magnitudes can be provided. A profile will be interpolated without constraint if the vector supplied for it is the the zero vector.

Ruled Skinning

Ruled skinning accepts the enumerated type “ruled_skinning”. This enumerated type has only one constant: “ruled”. This variation of skinning interpolates the profiles by creating a ruled surfaces for every pair of subsequent wires. Ruled skinning accepts degenerate end points. In addition, it supports the closed option. It does not detect self intersections.

Branched Skinning

Branched skinning differs from all other forms of skinning in that it allows multiple lists of cross sections to be skinned. The first and second parameters pertain to the “trunk” of the skin. Although the order of cross sections is not important in regular skinning, it is important here that the last cross section of the trunk be the section in which each of the branches will attach. Parameters three, four and five pertain to the branches of the skin. Branched skinning can handle any number of branches but most practical applications will only require a few. Like the trunk, the order of the cross sections in each branch is important. The first cross section must be the section that attaches to the trunk.

Each branch and trunk must contain at least one profile. There must be at least one branch.

In addition, this functional variant includes the enumerated type skinning_normals. It does not support degenerate end points.

Special Options

The api_skin_wires function accepts the following special options depending on the specific API:

The *arc_length* option is used to choose arc length or isoparametric parameterization of the skinning surfaces. For isoparametric parameterization, the surface parameter in the *V* direction follows the cross section curves. For arc length parameterization, the surface parameter follows lines of constant length. The default is isoparametric parameterization. Surfaces resulting from skinning with guide curves support isoparametric parameterization only. The default is FALSE.

The *no_twist* option may be used to minimize the twist of the surface produced. Twist minimization aligns closed curves such that the start of the second curve is aligned to the start of the first curve. Even if a body’s shape is unaffected by twisting, a surface with a twist could produce unexpected results when faceting and rendering. The default is TRUE.

The *align* option is used to align the direction of the cross section curves such that the normal of the first profile points towards the second profile. All other profiles are aligned to follow the first and second. If the sections are not oriented in the same direction, the align option should be used to avoid producing a twisted, self intersecting body. The default is TRUE.

The *simplify* option simplifies the surface to a conical surface, if applicable. If all of the cross sections lie on a conical surface (plane, cylinder, cone, sphere, or torus), the conical surface is created instead. The SPAresabs variable is used to determine whether or not the cross section lies on a conical surface. The default is TRUE.

The *closed* option may be used when the user needs to construct a solid body closed in *V.* (i.e., a torus). A solid body will be constructed only when all the wires supplied are closed. At least three profiles must be provided to create a closed body. The default is FALSE.

The *solid* option may be used when a solid loft must be constructed but a closed body is not desired. When a closed body is not desired, the end wires are capped with planar faces. The default is TRUE.

The *periodic* option allows the construction of loft bodies that are periodic in *V,* i.e., bodies that close back on themselves smoothly (continuously) at the start and end profiles. This option is activated in the skinning APIs by giving the *closed* option a value of 2. In Scheme, this is achieved by setting the periodic flag to TRUE. As for the closed option, at least three profiles must be supplied to create a periodic loft body. The default is FALSE.

The *virtual guide* option may be used in order to have the user defined guides affect the body in a global nature. The default is FALSE.

When the *merge_wirecoedges* option is set to TRUE, the G1 vertices of the skinning and lofting wire profiles are removed by merging adjacent coedges/edges. This improves operations such as blending and shelling because it reduces the coedge/edge count and the number of surfaces, and eliminates near tangent edges. The default is TRUE.

When the *estimate_loft_tanfacs* option is on, the weight factor for the tangency conditions of the loft will be determined such that it minimizes the average radius of curvature of the lofting surfaces. The resulting bodies should support shelling to greater thickness and also blending of their edges to larger blend radii. The default is FALSE.

The *match_vertices* option suppresses the vertex-matching-algorithm which ensures that all profiles consist of the same number of coedges. A heuristic approach is used to determine which vertex pairs are good matches. Profile coedges are then split where additional vertices are needed. This option is forced to TRUE if the coedge numbers of the profiles are not equal. Its default is TRUE.

The algorithm that minimizes the surface twist (see ”no_twist” option) may add vertices to some of the profiles if none of the existing vertices match well enough. The ”no_new_twist_vertices” option allows to force the algorithm to choose matching vertices from the existing vertices. The default is FALSE.

Errors: A wire pointer in the array is NULL or not to a wire body.

Limitations: The wire bodies are assumed to be simple; i.e., they must be well behaved and non-looping.

The parameterization of the surface in its U direction – the direction of the user-defined wire bodies – follows the parameterization of the curves underlying the wire bodies.

Skinning cannot produce a surface in which there is any one point on the surface in which the U and V directions are the same or opposite. That is, in no case can an underlying curve of the user-defined wire body and a lateral edge of the surface generated from the skinning algorithm ever meet tangentially.

End capping (for creation of open solid bodies) is supported only for planar end profiles. If the end capping operation fails then the sheet body constructed is returned.

The internal Booleans computed during the branched skinning operation are subject to the limitations of the Boolean engine.

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Changes model

api_skin_faces

Function: Skinning and Lofting, Creating Entities

Action: Unites two bodies using skinning between two faces.

Prototype:

```
outcome api_skin_faces (
    FACE* face1,                    // first face
    FACE* face2,                    // second face
    BODY*& body,                    // solid body
                                    // returned
    logical arc_length_option       // arc length
         = FALSE                    //
    logical twist_option = TRUE     // if the co-edges
                                    // are all closed,
                                    // minimize the
                                    // surface twist
    logical align_direction_option   // directions of
                                    // the co-edges, so
                                    // that they are the
                                    // same
    logical simplify_option         // simplify geometry
                                    // to conical surface
                                    // if applicable
    AcisOptions* ao = NULL      // ACIS options such
                                // as version/journal
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "kernel/kerndata/top/body.hxx"
#include "skin/kernapi/api/skinapi.hxx"
#include "skin/sg_husk/skin/skin_opts.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: If one the bodies is a solid, the operation is different than if it were a sheet body. In the case of a solid body, the face provided as an argument is removed from the body. Then, that face’s peripheral edges are used to skin to the peripheral edges of the second face. The result is one body.

This does not check to see if the resulting body is self-intersecting. Skinning does not use take-off vectors of the associated faces.

When ACIS makes a BODY from a FACE using api_skin_faces, it makes a single-sided face body. A single-sided face body is really a solid, not a sheet (infinitely thin) body. When a face body is single-sided, basic solid modeling concepts (and ACIS) consider the body to be a solid which extends from the back side of the face out to infinity with ill defined boundaries extending where the edges of the original face extend backward. Subsequent operations such as Booleans may not work, depending on how the single-sided face body is being used.

Errors:	None
Limitations:	None
Library:	skin
Filename:	skin/skin/kernapi/api/skinapi.hxx
Effect:	Changes model

api_start_vertex_sli

Function: Skinning and Lofting

Action: Modifies which vertex in a loop of coedges forming a wire is the starting vertex for traversing the loop.

Prototype:

```
outcome api_start_vertex_sli (
    AcisSLInterface* obj,   // AcisSLInterface object
    WIRE* wire,             // wire that contains
                            // the vertex
    VERTEX* vertex,         // becomes new starting
                            // vertex
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

```
outcome api_start_vertex_sli (
    AcisSLInterface* obj,   // AcisSLInterface object
    int index,              // index of wire which
                            // contains the vertex in
                            // lofting Interface
                            // object
    VERTEX* vertex,         // becomes new starting
                            // vertex
    AcisOptions* opts = NULL// ACIS options such
                            // as version/journal
    );
```

Includes:	`#include ”kernel/acis.hxx”`
	`#include ”kernel/kernapi/api/api.hxx”`
	`#include ”kernel/kerndata/top/vertex.hxx”`
	`#include ”kernel/kerndata/top/wire.hxx”`
	`#include ”skin/kernapi/api/skinapi.hxx”`
	`#include ”skin/sg_husk/skin/sur_intr.hxx”`
	`#include ”kernel/kernapi/api/acis_options.hxx”`

Description:	Refer to Action.
Errors:	None
Limitations:	None
Library:	skin
Filename:	skin/skin/kernapi/api/skinapi.hxx skin/skin/sg_husk/api/loft.cxx
Effect:	Changes model

api_terminate_skinning

Function:	Modeler Control, Skinning and Lofting
Action:	Terminates the skinning library.
Prototype:	`outcome api_terminate_skinning ();`

Includes:	`#include ”kernel/acis.hxx”`
	`#include ”kernel/kernapi/api/api.hxx”`
	`#include ”skin/kernapi/api/skinapi.hxx”`

Description:	Refer to Action.
Errors:	None
Limitations:	None
Library:	skin
Filename:	skin/skin/kernapi/api/skinapi.hxx
Effect:	System routine

api_valid_start_vertices_sli

Function:	Skinning and Lofting
Action:	Gets a list of valid starting vertices for skinning or lofting.

Prototype:	see below

Prototype:

```
outcome api_valid_start_vertices_sli (
    AcisSLInterface* obj,    // AcisSLInterface object
    WIRE* wire,              // wire to examine
    ENTITY_LIST& vertex_list,// list of valid
                             // vertices
    AcisOptions* opts = NULL // ACIS options such
                             // as version/journal
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kerndata/top/wire.hxx”
#include ”skin/kernapi/api/skinapi.hxx”
#include ”skin/sg_husk/skin/sur_intr.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: skin

Filename: skin/skin/kernapi/api/skinapi.hxx

Effect: Read–only