Chapter 16.
Functions Aa thru Az

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.

angle_between

Function: Mathematics, Analyzing Models

Action: Gets the angle (in radians) between two vectors or two unit vectors in the range 0 <= angle < 2 pi.

Prototype:
```
double angle_between (
    const SPAunit_vector& v1,   // first vector
    const SPAunit_vector& v2,   // second vector
    const SPAunit_vector& z     // normal to plane
        =*(SPAunit_vector*)NULL_REF
    );

double angle_between (
    const SPAvector& v1,        // first vector
    const SPAvector& v2,        // second vector
    const SPAunit_vector& z  // normal to plane
        =*(SPAunit_vector*)NULL_REF
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/geomhusk/geom_utl.hxx"
#include "baseutil/vector/unitvec.hxx"
#include "baseutil/vector/vector.hxx"
```

Description: This function is overloaded and can accept two vectors or two unit vectors as arguments.

The third vector is the plane in which the angle is measured, and is required. It also defines a direction, from the first to the second vector. To simply get the angle in 3–space, pass the normalized cross product of the two input vectors as the third vector.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/geomhusk/geom_utl.hxx

Effect: Read–only

api_abort_state

Function: History and Roll

Action: Deletes the current delta state and rolls model to the state before the current state.

Prototype:
```
outcome api_abort_state (
    HISTORY_STREAM* hs      // use default stream
        = NULL              // if NULL
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/bulletin/bulletin.hxx”
```

Description: Deletes the current delta state rolling the model to the state before construction of the current state was started.

If no stream is supplied, the default stream is used.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_add_state

Function: History and Roll

Action: Merges a DELTA_STATE instance into a HISTORY_STREAM.

Prototype:
```
outcome api_add_state (
    DELTA_STATE* ds,        // state to add
    HISTORY_STREAM* hs      // stream to add
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: This API grafts a DELTA_STATE into a HISTORY_STREAM following the active DELTA_STATE of the stream. This is used to in conjunction with api_note_state and api_remove_state to build multiple independent history streams. After noting a state, it can be moved the an alternate stream by removing it from the default stream, with api_remove_state, and adding it to the stream it is to become a part of.

Errors: Either input pointer is NULL.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_alternating_keep_pattern

Function: Patterns

Action: Creates a new pattern by applying an alternating keep–filter to an existing pattern.

Prototype:
```
outcome api_alternating_keep_pattern (
    pattern*& pat,          // created pattern
    const pattern& in_pattern,// input pattern
    logical keep1,          // 1st keep value
    logical keep2,          // 2nd keep value
    int which_dim,          // dimension for filter
    logical merge = TRUE,   // merge or replace flag
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernutil/law/pattern.hxx”
#include ”kernel/kernutil/law/pattern_api.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description:

Applies an alternating keep–filter to an existing pattern, merging with any existing filter or, optionally (with merge = FALSE), replacing it. The arguments keep1 and keep2 are successive Boolean keep values. which_dim specifies the dimension in which the filter is applied.

The following code snippet gives an example of using this API.

```
// Create a pattern
pattern* pat = NULL;
SPAvector x_vec(4.0, 0, 0);
int num_x = 8;
SPAvector y_vec(0, 2.0, 0);
int num_y = 10;
check_outcome(result = api_linear_pattern(pat, x_vec,
num_x,  y_vec, num_y));

// Modify the pattern
pattern* mod_pat = NULL;
logical keep1 = FALSE;
logical keep2 = TRUE;
int which_dim = 1;
check_outcome(result =
api_alternating_keep_pattern(mod_pat,  *pat, keep1,
keep2, which_dim));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad,  num_sides, prism));

// Apply the pattern to the prismcheck_outcome(result
= api_set_entity_pattern(prism, mod_pat));

// Clean up
pat–>remove();
mod_pat–>remove();
```

Errors: The keep is NULL, the period is less than one, or the specified dimension is not consistent with the pattern dimensionality.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_alternating_scale_pattern

Function: Patterns

Action: Creates a new pattern by applying an alternating scale to an existing pattern.

Prototype:

```
outcome api_alternating_scale_pattern (
    pattern*& pat,           // pattern returned
    const pattern& in_pattern,// input pattern
    double scale1,           // 1st scale value
    double scale2,           // 2nd scale value
    int which_dim,           // dimension for scaling
    const SPAposition& root, // position about which
                             // scaling is applied
    logical merge = TRUE,    // merge or replace flag
    AcisOptions* ao = NULL   // acis options
    );

outcome api_alternating_scale_pattern(
    pattern*& pat,               // pattern returned
    const pattern& in_pattern,   // input pattern
    const SPAvector& scale1,     // 1st scale value
    const SPAvector& scale2,     // 2nd scale value
    int which_dim,               // scaling dimension
    const SPAposition& root,     // position about
                                 // which scaling is
                                 // applied
    logical merge = TRUE,        // merge/replace flag
    AcisOptions* ao = NULL       // acis options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernutil/law/pattern.hxx”
#include ”kernel/kernutil/law/pattern_api.hxx”
#include ”baseutil/vector/vector.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description:

Applies an alternating scale to an existing pattern, merging with any existing scaling or, optionally (with merge = FALSE), replacing it. The arguments scale1 and scale2 give the alternating scale values, and can be given as vectors when nonuniform scaling is desired. which_dim specifies the dimension in which the scale is applied. The position root specifies the neutral point about which the scaling takes place (i.e., the point on the seed entity that remains fixed while the entity’s dimensions are altered). All scale values must be greater than zero.

The following code snippet gives an example of using this API.

```
// Create a pattern
pattern* pat = NULL;
SPAvector x_vec(4.0, 0, 0);
int num_x = 8;
SPAvector y_vec(0, 2.0, 0);
int num_y = 10;
check_outcome(result = api_linear_pattern(pat, x_vec,
num_x,  y_vec, num_y));

// Modify the pattern
pattern* mod_pat = NULL;
double scale1 = 0.8;
double scale2 = 1.2;
int which_dim = 1;
SPAposition root(0, 0, 0);
check_outcome(result =
api_alternating_scale_pattern(mod_pat,    *pat,
scale1, scale2, which_dim, root));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad,  num_sides, prism));
```

```
// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
mod_pat));

// Clean up
pat->remove();
mod_pat->remove();
```

Errors: A scale value is negative or zero.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_apply_transf

Function: Transforms, Modifying Models

Action: Changes the transform entity attached to a body.

Prototype:
```
outcome api_apply_transf (
    ENTITY* entity,          // entity to get new
                             // transform
    SPAtransf const& trans,  // new transform
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "baseutil/vector/transf.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: When transformations are applied to a body in ACIS, the underlying geometries of all the subordinate entities are not changed. This API simply attaches the transformation to the body entity and indicates that calculations should pipe the geometry through the transform. Each body's transformation matrix gives the relationship between its internal coordinate system and the coordinate system of the world.

If you want the transform actually applied to the geometry, use the api_change_body_trans function. One way is to apply the transformation first:

```
BODY* my_body;
api_apply_transf(my_body,  transf);
```

Then change the geometry of the object according to the transformation and set the body's transform to an empty transformation. (This does increase the risk of introducing round–off errors to the geometry.)

```
api_change_body_trans(my_body,NULL);
```

Use transformations with caution. Scaling and translation effects can combine to produce increasingly severe gaps in the geometry. Scaling transforms not only scale up or down the geometry, but also scale up or down gaps in the geometry. If you translate the geometry, you can move it far enough away from the origin that a gap is represented with 0 bits of resolution, and you cannot resolve it. Since SPAresabs doesn't change, at some point geometric operations fail.

Errors: The pointer to an entity is NULL.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_bb_begin

Function: History and Roll

Action: Starts the API bulletin board.

Prototype:
```
void api_bb_begin (
    logical linear          // linear or distributed
        = TRUE              // history stream
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "baseutil/logical.h"
```

Description: This API function is not intended to be called directly by the application, but rather via the API_BEGIN macro.

This routine may be used with api_bb_end to bracket a sequence of API calls so that they produce a single bulletin board. Its effect is cumulative so that when there are nested calls to api_bb_begin and api_bb_end, only the outermost pair of calls takes effect. In this way a new API routine may call existing API routines and appears to the caller like any other API routine in its handling of bulletin boards.

It should normally be called with an argument of TRUE, but if called with FALSE, the current bulletin–board (if any) is ”stacked”, and a new one started anyway. The corresponding api_bb_end rolls back and deletes this bulletin board, and reinstates the stacked one for more changes.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/api.hxx

Effect: System routine

api_bb_delete

Function: History and Roll

Action: Deletes bulletin boards.

Prototype:
```
void api_bb_delete ();
```

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

Description: This API is not intended to be called directly by the application. If a current bulletin board exists and has been ended and marked as unsuccessful, this function rolls back the model by undoing the changes recorded in the bulletin board, and then deletes the bulletin board (so freeing up the space occupied by old versions of records).

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/api.hxx

Effect: System routine

api_bb_end

Function: History and Roll

Action: Terminates the API bulletin board.

Prototype:
```
void api_bb_end (
    outcome& result,         // outcome
    logical linear           // linear or distributed
        = TRUE,              // history stream
    logical delete_stacked_bb   // deleted a stacked
     = FALSE                 // bulletin board
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "baseutil/logical.h"
```

Description: This API function is not intended to be called directly by the application, but rather via the API_END macro.

It is used with api_bb_begin to bracket a sequence of API calls so that they produce a single bulletin board. Its effect is cumulative so that when there are nested calls to api_bb_begin and api_bb_end, only the outermost pair of calls takes effect. In this way a new API routine may call existing API routines and appears to the caller like any other API routine in its handling of bulletin boards. It should normally be called with the second argument true.

Provided option logging is on and a bulletin board is already being constructed and it matches the initial call to api_bb_begin, this routine ends the current bulletin board, setting the success or not as recorded in the given outcome, into the bulletin board, and setting a reference to the bulletin board into the outcome. It should normally be called with the second argument true.

It then decrements a flag to say that a bulletin board is being constructed (unless the second argument is false).

If delete_stacked_bb is TRUE, a stacked bulletin board that results from a successful API_TRIAL block will be deleted.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/api.hxx

Effect: System routine

api_calculate_edge_tolerance

Function: Precision and Tolerance, Tolerant Modeling

Action: Calculates the tolerance of an edge.

Prototype:
```
outcome api_calculate_edge_tolerance (
    EDGE* edge,              // edge to test
    double& tol,             // resulting tolerance
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/top/edge.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This function calculates the tolerance of an EDGE or a TEDGE and returns a tolerance value. It does not use the tolerance value on the TEDGE.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_calculate_vertex_tolerance

Function: Precision and Tolerance, Tolerant Modeling

Action: Calculates the tolerance of a vertex.

Prototype:
```
outcome api_calculate_vertex_tolerance (
    VERTEX* vertex,          // input vertex / tvertex
    double& tol,             // resulting tolerance
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/top/vertex.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This function calculates the tolerance of a VERTEX or a TVERTEX and returns a tolerance value. It does not use the tolerance value on the TVERTEX.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_change_body_trans

Function: Transforms, Modifying Models

Action: Substitutes the given transform for the existing transform of the body.

Prototype:
```
outcome api_change_body_trans (
    BODY* body,                  // body to get new
                                 // transform
    TRANSFORM* new_transform,    // new transform
    logical negate = FALSE,      // negate the body
    AcisOptions* ao = NULL       // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/geom/transfrm.hxx"
#include "kernel/kerndata/top/body.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API changes the geometry to leave the body unaltered in its global coordinate space. Each body contains a transformation matrix that gives the relationship between the internal coordinate system and that of the world coordinate system. This API transforms the geometric definitions within the object so that with the new transformation set in the body transformation, the shape and position of the object are unchanged.

If negate is TRUE, this API negates the body by reflecting it about the origin and reversing all directions.

Calling this API with a NULL transform pointer leaves the body with a NULL transform, and any existing transforms are applied to the body geometry. For example, to scale the body's geometry, first call this API with the scaling transform and then call it again with a NULL transform.

Use transformations with caution. Scaling and translation effects can combine to produce increasingly severe gaps in the geometry. Scaling transforms not only scale up or down the geometry, but also scale up or down gaps in the geometry. If you translate the geometry, you can move it far enough away from the origin that a gap is represented with 0 bits of resolution, and you cannot resolve it. Since SPAresabs doesn't change, at some point geometric operations fail.

Call api_change_body_trans after api_transform_entity.

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

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_change_state

Function: History and Roll

Action: Modifies the modeler state by applying a delta state.

Prototype:
```
outcome api_change_state (
    DELTA_STATE* ds          // delta state to be
                             // applied
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: This API modifies the modeler's state to a different state using the given delta state. For example, the delta state carries the modeler from state *A* to state *B* and is applied only when the modeler is in state *A*.

Errors: The pointer to ds is NULL.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_change_to_state

Function: History and Roll

Action: Modifies the modeler state by applying zero or more delta_states.

Prototype:
```
outcome api_change_to_state (
    HISTORY_STREAM* hs,      // history state to be
                             // applied
    DELTA_STATE* ds,         // delta state to be
                             // applied
    int& n_actual            // Number of delta states
                             // rolled returned
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/bulletin/bulletin.hxx”
```

Description: This API modifies the modeler’s state to match that when the given delta state was first noted. The system finds the appropriate path through the history stream of which the delta state is a member.

Errors: The pointer to the delta state is NULL.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_checking

Function: Debugging

Action: Sets the pointer argument checking for an API call to on or off.

Prototype:
```
outcome api_checking (
    logical on_off           // TRUE for on
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "baseutil/logical.h"
```

Description: With argument checking on, pointer arguments to an API are tested to determine whether they are NULL. If they are NULL, a message is printed and the API returns an outcome with a nonzero error code.

Checks are also made on certain distances and angles supplied to APIs. Some APIs make more extensive checks internally, but the effect is the same. When there is an on error a message prints and the API returns an outcome with a nonzero error code.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_check_edge_errors

Function: Debugging, Tolerant Modeling

Action: Checks whether edges have errors that require them to be made tolerant, and optionally performs this conversion.

Prototype:
```
outcome api_check_edge_errors (
    ENTITY_LIST const& edges,   // input edges
    ENTITY_LIST& bad_edges,     // bad edge list
    ENTITY*& worst_entity,      // worst entity
    double& worst_error,        // worst error
    double tol                  // given tolerance
        = SPAresabs,
    logical stop_immediately    // if TRUE, stop
        = FALSE,                // after first bad
                                // edge is found
    ENTITY_LIST& new_edges      // tolerant edges
        =*(ENTITY_LIST*)NULL_REF,
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Checks the edges in the given list for gaps worse than the specified tolerance. Any such edges are added to the bad_edges list. If new_edges is given, such edges are converted into tolerant edges, and the end vertices are converted to tolerant vertices if necessary.

The bad_edges and new_edges lists are mapped so that bad_edges[i] is converted into new_edges[i].

If the stop_immediately flag is TRUE, processing stops after the first bad edge is found.

The worst_entity and worst_error always get set, even if the error in question was sufficiently small that the entity reported is not actually ”bad”.

Note that api_check_edge_errors normally converts ”bad” edges into tolerant ones. This function is only needed to check for ”bad” edges where none of the adjacent edges there needed to be made tolerant.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_check_face_loops

Function: Debugging, Model Topology

Action: Checks a face to see that it contains valid loops.

Prototype:
```
outcome api_check_face_loops (
    FACE* in_face,          // face to test
    int ai_info[]           // where test results
        = NULL,             // stored
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This function checks that the direction of a face's loops are correct. It eliminates the need to calculate the area of a face to determine the validity of the face. (If the area calculation for a face was negative, it was indicative of a problem usually in the direction of loops.)

This API returns outcome to indicate if the input face contains invalid loops. An error message is contained in the outcome.

ai_info[0]: number of periphery loops
ai_info[1]: number of holes
ai_info[2]: number of u separation loops
ai_info[3]: number of v separation loops
ai_info[4]: number of unknown loops
ai_info[5]: contains useful information

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_check_histories

Function: History and Roll

Action: Checks all HISTORY_STREAMs for problems.

Prototype:
```
outcome api_check_histories (
    HISTORY_STREAM_LIST* insane_list// list of
        = NULL,             // questionable streams
    FILE* fptr              // file for
        = stdout            // check output
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: Checks all HISTORY_STREAMs for mixing and improper entity IDs. Problems are reported to fptr, standard output by default, and HISTORY_STREAMs with errors are returned in the insane_list, if non-NULL.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_check_on

Function: Debugging

Action: Determines the status of checking and returns TRUE if it is on; otherwise, it returns FALSE.

Prototype:
```
logical api_check_on ();
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "baseutil/logical.h"
```

Description: Used with set_api_checking.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/api.hxx

Effect: System routine

api_check_vertex_errors

Function: Debugging, Tolerant Modeling

Action: Checks the vertices in the given list for gaps worse than the specified tolerance.

Prototype:

```
outcome api_check_vertex_errors (
    ENTITY_LIST const& vertices,// input vertex list
    ENTITY_LIST& bad_vertices,  // bad vertex list
    ENTITY*& worst_entity,      // worst entity
    double& worst_error,        // worst error
    double tol                  // given tolerance
        = SPAresabs,
    logical stop_immediately    // if TRUE, stop
        = FALSE,                // after first bad
                                // vertex is found
    ENTITY_LIST& new_vertices   // tolerant vertices
        =*(ENTITY_LIST*)NULL_REF,
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:

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

Description: Checks the vertices in the given list for gaps worse than the specified tolerance. Any such vertices are added to the bad_vertices list. If new_vertices is given, such edges are converted into tolerant edges, and the end vertices are converted to tolerant vertices if necessary.

The bad_vertices and new_vertices lists are mapped so that bad_vertices[i] is converted into new_vertices[i].

If the stop_immediately flag is TRUE, processing stops after the first bad vertex is found.

The worst_entity and worst_error always get set, even if the error in question was sufficiently small that the entity reported is not actually "bad."

api_check_edge_errors normally converts "bad" vertices into tolerant ones. This function is only needed to check for "bad" vertices where none of the adjacent edges needed to be made tolerant.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_clear_annotations

Function: Feature Naming

Action: Clears annotation entities from the currently active bulletin board.

Prototype:
```
outcome api_clear_annotations (
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Searches the global list of annotations and loses them. This should be called at some point after a modeling operation, such as sweeping or blending. Once the annotation information has been handled, it must be cleared from the active bulletin board using api_clear_annotations before the next modeling operation. Ideally, the operation to be annotated should be wrapped in an API_BEGIN/END block so the call to api_clear_annotations will restore the bulletin board to a state as if annotations had never been created.

Not calling api_clear_annotations can lead to a bloated bulletin board as well as incorrect links between separate modeling operations when option unhook_annotations is FALSE.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_copy_body

Function: Model Geometry, Model Object

Action: Creates a copy of a body.

Prototype:
```
outcome api_copy_body (
    BODY* body,              // body to be copied
    BODY*& new_body,         // copy returned
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: Given a body as input, copies the given body and all its associated (“connected”) entities, if any, using each entity’s copy and fix-up methods (e.g., copy_scan, copy_data, fix_pointers, etc.). This includes entities that are above and/or below the given body in the topological hierarchy. For example, copying an edge copies the coedges, loops, faces, shells, etc., as well as all the associated curves, vertices, points, attributes, etc. If there are no associated entities, only the given body is copied.

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

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_copy_entity

Function: Model Geometry, Model Object

Action: Creates a copy of an entity and all its associated entities.

Prototype:
```
outcome api_copy_entity (
    ENTITY* entity,          // entity to be copied
    ENTITY*& new_entity,     // copy returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/data/entity.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API copies the given entity and all its associated (“connected”) entities, if any, using each entity’s copy and fix-up methods (e.g., copy_scan, copy_data, fix_pointers, etc.). This includes entities that are above and/or below the given entity in the topological hierarchy. For example, copying an edge copies the coedges, loops, faces, shells, etc., as well as all the associated curves, vertices, points, attributes, etc. If there are no associated entities, only the given entity is copied.

Errors: The NULL pointer is given to entity.

Limitations: Refer to description.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_copy_entity_contents

Function: Model Geometry, Model Object

Action: Creates a copy of a topological entity all its associated subentities.

Prototype:
```
outcome api_copy_entity_contents (
    ENTITY* in_ent,          // entity to be copied
    ENTITY*& copy,           // copy returned
    SPAtransf& tr            // optional
        =*(SPAtransf*)NULL_REF,//  transformation
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″baseutil/vector/transf.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/data/entity.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API copies the given entity and all its associated subentities, if any. Subentities are those that are **below** the given entity in the topological hierarchy. It does not copy entities that are above the given entity. The optional transformation is applied to the copied entity, if applicable.

Note *This special-case function only operates on* VERTEX, EDGE, COEDGE, WIRE, LOOP, FACE, SHELL, *and* LUMP *entities; for all other entity types, it calls* api_copy_entity.

Errors: The pointer to an original entity is NULL

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_copy_entity_list

Function: Model Geometry

Action: Creates a copy of all entities in an entity list and all their associated entities.

Prototype:
```
outcome api_copy_entity_list (
    ENTITY_LIST& entity_list,   // list to copy
    ENTITY_LIST& copied_entity_list,// copy returned
    AcisOptions* ao = NULL      // acis options
    );
```

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

Description: This API copies the entities in the given entity list and all their associated ("connected") entities, if any, using each entity's copy and fix-up methods (e.g. , copy_scan, copy_data, fix_pointers, etc.). This includes entities that are above and/or below the given entity in the topological hierarchy. For example, copying an edge copies the coedges, loops, faces, shells, etc., as well as all the associated curves, vertices, points, attributes, etc. If there are no associated entities, only the given entities are copied. The returned entity list's entities are in the same order as the given entity list.

Errors: The entity_list is empty.

Limitations: Refer to description.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_create_history

Function: History and Roll

Action: Returns a newly created HISTORY_STREAM on the heap.

Prototype:
```
outcome api_create_history (
    HISTORY_STREAM*& hs     // created history stream
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_curve_pattern

Function: Patterns

Action: Creates a pattern parallel to a curve.

Prototype:

```
outcome api_curve_pattern (
    pattern*& pat,           // pattern returned
    const curve& in_curve,   // guiding curve
    const SPAinterval& param_range, // range
    int num_elements,        // number of elements
                             // in the pattern
    const SPAposition& root, // position mapped
                             // to the pattern sites
    logical on_endpoints     // flag for beginning and
        = FALSE,             // ending on endpoints
    law* rail_law            // rail law
        = NULL,              // to follow
    const SPAvector& rail_dir// direction mapped
        =*(SPAvector*)NULL_REF,// to rail direction
    const SPAvector& tangent_dir// direction mapped
        =*(SPAvector*)NULL_REF,// to tangent
                             // direction
    const SPAtransf& in_trans// check for
        =*(SPAtransf*)NULL_REF,// transform
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "baseutil/vector/interval.hxx"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/transf.hxx"
#include "baseutil/vector/vector.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerngeom/curve/curdef.hxx"
#include "kernel/kernutil/law/pattern.hxx"
#include "kernel/kernutil/law/pattern_api.hxx"
#include "lawutil/law_base.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Creates a one–dimensional pattern of number elements, equally spaced in parameter space, along the curve specified by the in_curve argument, over the interval given by param_range. The argument root specifies the position (which can be on or off the pattern seed entity, as desired) to be mapped to the pattern sites. The pattern can be extended to the endpoints of the edge by setting on_endpoints to TRUE. By default, pattern members are oriented identically to one another. They will follow a rail law if rail_law is provided. In that case, the vectors rail_dir and tangent_dir specify the directions, relative to the seed entity, that are mapped to the rail law and tangent directions of the edge.

The following code snippet gives an example of using this API.

```
// Create a spiral curve
EDGE* edge = NULL;
SPAposition center(0, 0, 0);
SPAvector normal(0, 0, 1);
SPAposition start_position(3, 0, 0);
double width = 3.0;
double angle = 6.0 * M_PI;
check_outcome(result = api_edge_spiral(center,
normal,     start_position, width, angle, edge));
const curve& crv = edge->geometry()->equation();
SPAinterval param_range = edge->param_range();
if (edge->sense() == REVERSED) param_range.negate();

// Create a pattern
pattern* pat = NULL;
int number = 36;
SPAposition root(0, 0, 0);
check_outcome(result = api_curve_pattern(pat, crv,
           param_range, number, root));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad,  num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
pat));
```

```
// Clean up
pat->remove();
check_outcome(result = api_del_entity(edge));
```

Errors: The number of elements is less than one, or the rail direction was specified without specifying a tangent direction.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_cylindrical_pattern

Function: Patterns

Action: Creates a pattern with cylindrical symmetry.

Prototype:

```
outcome api_cylindrical_pattern (
    pattern*& pat,           // created pattern
    const FACE* in_face,     // face defining pattern
                             // axis and radius
    int num_angular,         // # of pattern elements
                             // about cylinder axis
    int num_axial            // # of pattern elements
        = 1,                 // along cylinder axis
    double ring_spacing      // distance between
        = 0.0,               // circular pattern
                             // layers
    logical alternating      // flag to stagger angle
        = FALSE,             // between layers
    AcisOptions* ao = NULL   // acis options
    );
```

```
outcome api_cylindrical_pattern (
    pattern*& pat,           // created pattern
    const SPAposition& center,// starting position
for
                             // cylinder axis
    const SPAvector& normal,// direction of the
                             // cylinder axis
    int num_angular,         // # of pattern elements
                             // about cylinder axis
    int num_axial            // # of pattern elements
        = 1,                 // along cylinder axis
    double ring_spacing      // distance between
        = 0.0,               // circular pattern
                             // layers
    logical alternating      // flag to stagger angle
        = FALSE,             // between layers
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include ″kernel/acis.hxx″
#include ″baseutil/logical.h″
#include ″baseutil/vector/position.hxx″
#include ″baseutil/vector/vector.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/face.hxx″
#include ″kernel/kernutil/law/pattern.hxx″
#include ″kernel/kernutil/law/pattern_api.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: Creates a two–dimensional pattern with cylindrical symmetry, with a radius and axis defined either by the center position and normal vector or by the cylindrical face in_face. The numbers of angular and axial elements in the pattern are set by num_angular and num_axial, respectively, and the distance between circular pattern layers by the spacing argument. If alternating is TRUE, adjacent layers are staggered in angle. The pattern coordinates are specified in the order (angular, axial).

The following code snippet gives an example of using this API.

```
// Create a pattern
pattern* pat = NULL;
SPAposition center(5, 0, 0);
SPAvector normal(0, 1, 0);
int num_angular = 8;
int num_axial = 5;
double spacing = 5.0;
check_outcome(result = api_cylindrical_pattern(pat,
center, normal, num_angular, num_axial, spacing));

// Create a cylinder
BODY* cylinder = NULL;
SPAposition bottom(0, 0, 0);
SPAposition top(0.5, 0, 0);
double maj_rad = 1.0;
double min_rad = 0.5;
check_outcome(result =
api_solid_cylinder_cone(bottom, top, maj_rad,
min_rad, maj_rad, NULL, cylinder));

// Apply the pattern to the prism
check_outcome(result =
api_set_entity_pattern(cylinder, pat));

// Clean up
pat->remove();
```

Errors: The number of angular or axial elements is less than one, or the face that is specified is not cylindrical.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_deep_copy_entity

Function: Model Geometry, Model Object

Action: Creates a deep copy of an entity and all its associated entities.

Prototype:
```
outcome api_deep_copy_entity (
    ENTITY* entity,          // entity to copy
    ENTITY*& new_entity,     // deep copy returned
    logical dpcpy_skip       // flag to skip
        = FALSE,             // attributes not
                             // deep–copyable
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API deep copies the given entity and all its associated (connected) entities, if any, using each entity's copy and fix-up methods (e.g., copy_scan, copy_data, fix_pointers, etc.). The difference between a deep copy and a regular copy is that a regular copy may contain references to shared underlying associated entities, but a deep copy will not. This includes entities that are above and/or below the given entity in the topological hierarchy. For example, deep copying an edge deep copies the coedges, loops, faces, shells, etc., as well as all the associated curves, vertices, points, attributes, etc. If there are no associated entities, only the given entity is deep copied.

Errors: Attempting to copy an entity that has associated entities that do not support a deep copy routine. The NULL pointer is given to the entity.

Limitations: Refer to description.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_deep_copy_entity_list

Function: Model Geometry

Action: Creates a deep copy of all entities in an entity list and all their associated entities.

Prototype:
```
outcome api_deep_copy_entity_list (
    ENTITY_LIST& entity_list,   // entities to copy
    ENTITY_LIST& new_entity_list,// copies returned
    logical dpcpy_skip      // flag to skip
        = FALSE,            // attributes not
                            // deep-copyable
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API deep copies the entities in the given entity list and all their associated (connected) entities, if any, using each entity's copy and fix-up methods (e.g. , copy_scan, copy_data, fix_pointers, etc.). The difference between a deep copy and a regular copy is that a regular copy may contain references to shared underlying associated entities, but a deep copy will not. This includes entities that are above and/or below the given entity in the topological hierarchy. For example, deep copying an edge deep copies the coedges, loops, faces, shells, etc. , as well as all the associated curves, vertices, points, attributes, etc. If there are no associated entities, only the given entities are deep copied. The returned entity list's entities are in the same order as the given entity list.

Errors: Attempting to copy an entity that has associated entities that do not support a deep copy routine.

The entity_list is empty.

Limitations: Refer to Description.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_delent

Function: Model Topology

Action: Deletes an entity and subentities, which are entities below the given entity in the topological hierarchy.

Prototype:
```
outcome api_delent (
    ENTITY* given_entity,    // entity to be deleted
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API deletes an entity and all entities below it in the topological hierarchy.

Errors: Pointer to entity is NULL or not to topology (BODY, LUMP, WIRE, etc.).

Limitations: Pointers above the deleted entity in the topological hierarchy are not guaranteed to be set to NULL. For example, when a lump is deleted, the body pointer to the lump may or may not be set to NULL. This function loses the given topological entity, all lower-level topological entities comprising the given entity, and reduces the use count. It could possibly remove any associated geometry. It does not affect any pointers that were pointing to any of the objects. When using this API, pointers that used to point to the entity need to be fixed, or the item could be unhooked and then deleted.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_delete_ds

Function: History and Roll

Action: Deletes a delta state and dependent data.

Prototype:
```
outcome api_delete_ds (
    DELTA_STATE* ds          // delta state to be
                             // deleted
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: This API deletes a delta_state; i.e., the recorded information that enables the modeler to change between two particular states.

Errors: NULL pointer to delta state.

Limitations: Delta states should be deleted starting with those furthest away and working toward the current state to ensure that delete bulletins (and rolled back create bulletins) are deleted last.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_delete_history

Function: History and Roll

Action: Deletes the HISTORY_STREAM and all ENTITYs in the stream.

Prototype:
```
outcome api_delete_history (
    HISTORY_STREAM* hs      // input history stream
        = NULL
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: Deletes the HISTORY_STREAM and all ENTITYs associated with BULLETIN on the HISTORY_STREAM. Therefore, no ENTITYs will be deleted when logging is off. Uses the default HISTORY_STREAM if none is supplied.

Errors: Fails when unable to remove all ENTITYs referred to in the stream.

Limitations: Logging must be used.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_del_entity

Function: Model Topology

Action: Deletes the given entity.

Prototype:
```
outcome api_del_entity (
    ENTITY* given_entity,    // entity to be deleted
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/data/entity.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API deletes an entity and all its associated entities. This loses all entities that are connected to the given entity. It will lose multiple bodies if they are connected by attributes. It does not affect any pointers that were pointing to any of the objects. This allows you to delete an entire entity from anywhere in the entity's topological hierarchy without having to traverse to the top of the topology chain.

Errors: Pointer to entity is NULL.

Limitations: Deletes entities above as well as below the specified entity in the hierarchy.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_del_entity_list

Function: Model Topology

Action: Deletes the given list of entities.

Prototype:
```
outcome api_del_entity_list (
    ENTITY_LIST& given_list, // entities to be deleted
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: This API deletes all the entities in an entity list and all their associated entities. This loses all entities that are connected to the given entities. It will lose multiple bodies if they are connected by attributes. It does not affect any pointers that were pointing to any of the objects. This allows deletion of an entire entity from anywhere in the entity's topological hierarchy without having to traverse to the top of the topology chain.

Use this API instead of api_del_entity when you need to delete more than one entity at a time, since calling api_del_entity repeatedly could be dangerous as the user has to keep track of what in the list has already been deleted.

Errors: None

Limitations: Deletes entities above as well as below the specified entities in the hierarchy.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_distribute_state_to_streams

Function: History and Roll

Action: Distributes BULLETINs in a DELTA_STATE to one or more HISTORY_STREAMs as directed by a StreamFinder.

Prototype:
```
outcome api_distribute_state_to_streams (
    DELTA_STATE* pState,         // delta state to be
                                 // distributed
    StreamFinder* pStreamFinder,// used to direct
                                 // distribution
    logical clearDelta,          // TRUE to delta
                                 // undistributed
                                 // bulletins
    logical hideStates           // TRUE to mark new
                                 // states as hidden
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/sg_husk/history/history.hxx"
#include "baseutil/logical.h"
```

Description:

This API distributes the given ds to one or more HISTORY_STREAMs as directed by the given StreamFinder. In each stream distributed to, a new DELTA_STATE will be created to hold the BULLETINs. StreamFinder is a class with one pure virtual function, findStream, which must return the HISTORY_STREAM* associated with the given entity. The findStream function may be called more than once for each entity. In a topology based search, the stream finder can cache data in an early pass, that can be used in a later pass. This is necessary because POINT, CURVE, PCURVE, and SURFACE do not know their owners. When the stream is found for the the corresponding VERTEX, EDGE, COEDGE and FACE, the stream for the subordinate entity can be saved and used in a later pass.

The base StreamFinder class provides functions for finding the stream based on an attached ATTRIB_HISTORY and for maintaining a mapping of entities to streams.

As an example, here is the StreamFinder used by the Part Management Component.

```
class StreamFinderPM : public StreamFinder {
    // A StreamFinder for the PM_HUSK.
    // Implements a nested approach to
    // distribution in which bulletins go to
    // the most specific stream available.
    // Part streams are more specific than
    // the default stream. Body streams are more
    // specific than part streams.
public:
    virtual HISTORY_STREAM* findStream( ENTITY* );
};
```

```
HISTORY_STREAM*
StreamFinderPM::findStream(
    ENTITY* pEntity
    )
{
    HISTORY_STREAM* pStream = NULL;
    // Look for a ATTRIB_HISTORY. If found add
    // the entity and associated geometry to
    // the stream map.
    pStream = findStreamFromAttribute(pEntity);

    if( !pStream ) {
        // Still no stream?.
        // Look for a stream on the part
        // the entity is in.
        PART* part = get_part(pEntity);
        if(part) {
            pStream = part->history_stream();
        }
    }

    if( pStream ) {
        addToStreamMap(pEntity, pStream);
    }

    return pStream;
}
```

The clearDelta argument tells how to handle BULLETINs for which a target stream could not be found. If TRUE they are deleted along with the input delta state. If FALSE, they are left in the input state.

The hideStates argument tells whether to mark the resulting states as hidden in the target streams. api_pm_roll_n_states does not count hidden states. Hidden states are useful for operations that should appear read only to the user. For example, a pick or display operation may calculate boxes and create DELTA_STATEs. One can hide these states so they are not apparent to the user.

Errors: The pointer to ds is NULL.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_edge_pattern

Function: Patterns

Action: Creates a pattern parallel to an edge.

Prototype:

```
outcome api_edge_pattern (
    pattern*& pat,          // created pattern
    COEDGE* in_coedge,      // coedge
    int number,             // number of elements
    const SPAposition& root,// start position
    logical on_endpoints    // extend to endpoints
        = FALSE,            // or not
    const SPAvector& normal_dir// use normal to
        =*(SPAvector*)NULL_REF,// edge face
    const SPAvector& tangent_dir// for rail law
        =*(SPAvector*)NULL_REF,
    AcisOptions* ao = NULL  // acis options
    );

outcome api_edge_pattern (
    pattern*& pat,          // created pattern
    EDGE* in_edge,          // edge
    FACE* in_face,          // face
    int number,             // number of elements
    const SPAposition& root,// start position
    logical on_endpoints    // extend to endpoints
        = FALSE,            // or not
    const SPAvector& normal_dir// use normal to
        =*(SPAvector*)NULL_REF,// edge face
    const SPAvector& tangent_dir// for rail law
        =*(SPAvector*)NULL_REF,
    AcisOptions* ao = NULL  // acis options
    );
```

```
outcome api_edge_pattern (
    pattern*& pat,          // created pattern
    EDGE* in_edge,          // edge
    int number,             // number of elements
    const SPAposition& root,// start position
    logical on_endpoints    // extend to endpoints
        = FALSE,            // or not
    const SPAvector& rail_dir// for rail law
        =*(SPAvector*)NULL_REF,
    const SPAvector& tangent_dir// for rail law
        =*(SPAvector*)NULL_REF,
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/vector.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/coedge.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "kernel/kernutil/law/pattern.hxx"
#include "kernel/kernutil/law/pattern_api.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description:

Creates a one–dimensional pattern of number elements, equally spaced in parameter space, parallel to the edge or coedge specified by the in_edge or in_coedge argument. The argument root specifies the position (which can be on or off the pattern seed entity, as desired) to be mapped to the pattern sites. The pattern can be extended to the endpoints of the edge by setting on_endpoints to TRUE. By default, the orientations of pattern members are identical. If in_edge alone is given, they will instead follow the edge's rail law if rail_dir and tangent_dir are specified; if in_face is also furnished, or if in_coedge is specified instead, they will follow the normal to the edge's face if normal_dir and tangent_dir are given.

The following code snippet shows an example of how this API can be used.

```
// Create a spline edge
EDGE* edge = NULL;
SPAposition pts[7];
pts[0] = SPAposition(0, 0, 0);
pts[1] = SPAposition(10, 5, 0);
pts[2] = SPAposition(20, 2, 0);
pts[3] = SPAposition(30, 8, 0);
pts[4] = SPAposition(40, 2, 0);
pts[5] = SPAposition(50, 5, 0);
pts[6] = SPAposition(60, 0, 0);
SPAunit_vector dir_start(0, 1, 0);
SPAunit_vector dir_end(0, –1, 0);
check_outcome(result = api_curve_spline(7, pts,
       &dir_start, &dir_end, edge));

// Create a pattern
pattern* pat = NULL;
int number = 20;
SPAposition root(0, 0, 0);
check_outcome(result = api_edge_pattern(pat, edge,
       number, root));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad,  num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
pat));

// Clean up
pat–>remove();
check_outcome(result = api_del_entity(edge));
```

Errors: The number of elements is less than one, or the normal (or rail) direction was specified without specifying a tangent direction, or a NULL entity was specified.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_elliptical_pattern

Function: Patterns

Action: Creates an elliptical pattern.

Prototype:
```
outcome api_elliptical_pattern (
    pattern*& pat,           // created pattern
    const SPAposition& center,// center of pattern
    const SPAvector& normal, // normal to pattern
                             // plane
    int num_elements,        // # of pattern elements
    logical not_rotate       // TRUE eliminates
        = FALSE,             // rotation of elements
    const SPAposition& root  // position mapped to
        =*(SPAposition*)NULL_REF,// pattern sites
    double angle             // angular extent
        = 2.0* 3.14159265358979323846,// of pattern
    double ratio             // ratio of major/minor
        = 1.0,               // radii
    const SPAvector& major_axis// orientation of
        =*(SPAvector*)NULL_REF,// major axis
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/vector.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernutil/law/pattern.hxx”
#include ”kernel/kernutil/law/pattern_api.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Creates a one–dimensional elliptical pattern defined by an axis of rotation. The center and normal arguments indicate the (global) position and orientation of the axis. The number argument defines the number of entities in the pattern. These elements are kept in a fixed relative orientation if not_rotate is TRUE, in which case root, the position that is mapped to the pattern sites, must be specified. The angle argument fixes the angular extent of the pattern, with positive or negative values indicating a pattern proceeding clockwise or counter–clockwise about the normal vector. The ratio argument sets the ratio of minor/major radii of the pattern. If major_axis is given, it specifies the major axis of the pattern; otherwise, this axis is directed from center to root.

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAposition center(10, 0, 0);
SPAvector normal(0, 0, 1);
int number = 12;
check_outcome(result = api_elliptical_pattern(pat,
center, normal, number));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
pat));

// Clean up
pat–>remove();
```

Errors: The number of elements is less than one, or the user failed to supply a root position with not_rotate set to TRUE.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_end_journal

Function: ACIS Journal

Action: Sets the status flag off journalizing and finishes the snapshot journaling mechanism.

Prototype:
```
outcome api_end_journal (
    AcisOptions* ao          // acis options such as
                             // version, journal
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/acis_journal.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernapi/api/api.hxx"
```

Description: Sets the status flag to off and writes down the script footer.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/acis_journal.hxx

Effect: System routine

api_ensure_empty_root_state

Function: History and Roll

Action: If necessary, adds an empty delta state to the beginning of the history stream so that users can roll to a state with no entities.

Prototype:
```
outcome api_ensure_empty_root_state (
    HISTORY_STREAM* history, // history stream to be
                             // modified
    DELTA_STATE*& root_state // pointer to the (empty)
                             // root delta state
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: This routine examines the root delta state of the specified history stream. If the root state is empty (no bulletin boards), then it does nothing. If the root state is non-empty, then it adds a new, empty, root state immediately ”before” the original root state. In either case, it returns (through the root_state argument) a pointer to the resulting empty root state.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_extract_coed_info

Function: Object Relationships

Action: Computes the given number of equidistant points in the parametric space of the underlying curve for the coedge.

Prototype:
```
outcome api_extract_coed_info (
    COEDGE* coedge,          // coedge of face
    logical forward,         // forward direction of
                             // evaluation
    logical outward,         // tangents point off of
                             // face
    int num_pts,             // size of arrays/number
                             // of points where to
                             // evaluate
    SPAposition* pts,        // points along edge
                             // returned (user
                             // allocates arrays)
    SPAunit_vector* tans     // surface tangents along
                             // edge at positions
                             // returned
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/top/coedge.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/unitvec.hxx”
```

Description: This API returns surface tangents perpendicular to the curve. Sense flag forward controls the direction for ordering of points. Sense flag outward indicates whether the face tangent points away from or into the face. This function is useful for interpolating surfaces to join with the face.

Errors: Entity NULL or not a coedge.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_find_annotations

Function: Feature Naming

Action: Finds all annotation entities of a given type.

Prototype:
```
outcome api_find_annotations (
    ENTITY_LIST&,           // list of annotation
                            // entities
    is_fun                  // test for specific
        = is_ANNOTATION,    // type of annotation
    BULLETIN_BOARD* bb      // obsolete, ignored
        = NULL,
    AcisOptions* = NULL     // acis options
    );
```

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

Description: When the annotations option is turned on, certain modeling operations like blending or sweeping produce ANNOTATION class instances in internal ENTITY_LIST. The api_find_annotations function can be used to acquire a list of those annotations for user–defined processing. Generally, annotations are cleared manually from the list by using api_clear_annotations before the next modeling operation.

The flag for is_fun defaults to is_ANNOTATION. However, any is_ function for a class can be used. So, for example, to get the top vertex annotations from a sweep operation, this function can be passed is_SWEEP_ANNO_VERTEX_TOP as an argument.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_find_named_state

Function: History and Roll

Action: Rolls to the start of a named state.

Prototype:
```
outcome api_find_named_state (
    const char* name,       // name of state to which
                            // to roll
    HISTORY_STREAM* hs,     // history stream to use
    DELTA_STATE_LIST& dslist// states found returned
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: This API find states in the stream with the given name and adds them to the given DELTA_STATE_LIST.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_get_active_entities

Function: History and Roll

Action: Finds all active entities associated with a history stream.

Prototype:
```
outcome api_get_active_entities (
    HISTORY_STREAM const* hs,   // stream to search
    ENTITY_LIST& ents,      // list into which active
                            // entities are placed
    logical unowned_only    // filter flag
        = FALSE,
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API finds all of the "active" entities associated with a history stream (i.e. created and not yet deleted in the main line). These entities are added to the entity list ents. Note that ents is not cleared by this routine, since this routine has "append to" syntax.

A TRUE unowned_only flag indicates that the user is only interested in a minimal set of highest level entities, typically a list of bodies. It filters out any entities which are not top–level, as well any points, curves, surfaces, transforms, annotations, or attributes found by scanning the remaining entities with a SCAN_DISTRIBUTE flag. (It ignores any entities which are owned.)

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_active_state

Function: History and Roll

Action: Returns the active DELTA_STATE in the given HISTORY_STREAM.

Prototype:
```
outcome api_get_active_state (
    DELTA_STATE*& active_ds, // returned delta state
    HISTORY_STREAM* hs       // input history stream
        = NULL
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: Returns the active DELTA_STATE in the given HISTORY_STREAM. The active DELTA_STATE is either the most recently closed state in the stream, made by calling note_state, or the state just rolled to. If no HISTORY_STREAM is supplied, the default stream is used.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_all_user_attributes

Function: History and Roll

Action: Finds all attributes of a specified user type in a history stream.

Prototype:
```
outcome api_get_all_user_attributes (
    HISTORY_STREAM const* hs,   // stream to search
    int derivation_level,    // number of levels
    int attrib_type_code,    // id of attrib type
    ENTITY_LIST& attribs,    // list to add to
    logical active_only      // backup type flag
        = TRUE,
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

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

Description: This routine is intended to allow users to find all of "their" attributes in a history stream. A user will typically pass in his master attribute type code to obtain a list all attributes in the history stream specific to his company. Any such attributes found are added to the attribs list. This routine does not clear attributes since it has "append to" syntax. derivation_level specified the number of levels of derivation of the requested attribute type from ENTITY.

The active_only flag is intended for use immediately after restoring the history stream from a file. If active_only is FALSE, then backup copies of the requested attribute type are also returned, allowing the user to perform direct post–restore operations that may be necessary to rebind these attributes to user data.

This flag setting violates the encapsulation of the roll mechanism; it should be used with great caution and alternative solutions (such as using entity IDs) should be explored. This flag setting is only intended to give users access to their own attributes. Passing in a Spatial type code when active_only is FALSE results in undefined behavior, possibly returning the error "access to non–user bulletin board entities is not allowed".

Errors: access to non–user bulletin board entities is not allowed

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_annotation_ctx

Function: Feature Naming

Action: Returns the annotation list.

Prototype:

```
outcome api_get_annotation_ctx (
    annotation_ctx*& list,   // Returns the
                             // annotation_ctx
    AcisOptions* = NULL      // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: When the annotations option is turned on, certain modeling operations like blending or sweeping produce ANNOTATION class instances stored in an annotation_ctx. This API returns a pointer to the annotation_ctx.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_get_coedges

Function: Model Topology

Action: Gets all the coedges related to an entity.

Prototype:
```
outcome api_get_coedges (
    ENTITY* ent,            // entity to examine
    ENTITY_LIST& coedge_list,// coedges related to
                            // entity returned
    PAT_NEXT_TYPE include_pat// how to treat
        = PAT_CAN_CREATE,   // patterned coedges
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/data/entity.hxx″
#include ″kernel/kerndata/lists/lists.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
#include ″kernel/kernutil/law/pattern_enum.hxx″
```

Description: If the input entity (ent) has COEDGEs; i.e., BODY, LUMP, FACE, etc., this API returns all COEDGEs of the entity.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – patterned objects are created if they do not already exist, and are included in the list.
PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.
PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_curve_ends

Function: Model Geometry, Construction Geometry

Action: Gets the end points of a curve.

Prototype:
```
outcome api_get_curve_ends (
    EDGE* crv,               // curve
    SPAposition& pt1,        // start position
                             // returned
    SPAposition& pt2,        // end position returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_default_history

Function: History and Roll

Action: Returns the default HISTORY_STREAM.

Prototype:
```
outcome api_get_default_history (
    HISTORY_STREAM*& default_hs // default history
                                // stream
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/bulletin/bulletin.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_edges

Function: Model Topology

Action: Gets all the edges related to an entity.

Prototype:
```
outcome api_get_edges (
    ENTITY* ent,             // entity to examine
    ENTITY_LIST& edge_list,  // edges related to
                             // entity returned
    PAT_NEXT_TYPE include_pat// how to treat
        = PAT_CAN_CREATE,    // patterned edges
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
#include ”kernel/kernutil/law/pattern_enum.hxx”
```

Description: If the input entity (ent) has EDGEs; i.e., BODY, LUMP, FACE, etc., this API returns all EDGEs of the entity. The input entity can also be a VERTEX; in which case, this API returns all EDGEs that share the common VERTEX.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – patterned objects are created if they do not already exist, and are included in the list.
PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.
PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_ellipse_parameters

Function: Model Geometry, Construction Geometry

Action: Gets the arguments for a circle or an ellipse.

Prototype:
```
outcome api_get_ellipse_parameters (
    EDGE* ell,                // ellipse or circle
    SPAposition& center,      // center returned
    SPAunit_vector& normal,   // normal to plane of
                              // ellipse returned
    SPAvector& major_axis,    // major axis returned
                              // (length equals major
                              // radius)
    double& radius_ratio,     // ratio of major radius
                              // to minor radius
                              // returned
    double& start_angle,      // start angle in radians
                              // returned
    double& end_angle,        // end angle in radians
                              // returned
    AcisOptions* ao = NULL    // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/unitvec.hxx”
#include ”baseutil/vector/vector.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

Errors: The curve is not an elliptical curve.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_entities

Function: Model Topology

Action: Gets all specified entities related to an entity.

Prototype:

```
outcome api_get_entities (
    ENTITY* ent,             // entity to examine
    ENTITY_LIST& ent_list,   // returned related
                             // entities
    ENTITY_ID topology_ids,  // topological selection
     ENTITY_ID geometry_ids,// geometrical selection
     PAT_NEXT_TYPE include_pat  // how to treat
        = PAT_CAN_CREATE,        // patterned faces
    AcisOptions* ao = NULL       // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernutil/law/pattern_enum.hxx"
#include "kernel/kerndata/top/alltop.hxx"
#include "kernel/kerndata/geom/allsurf.hxx"
#include "kernel/kerndata/geom/allcurve.hxx"
#include "kernel/sg_husk/query/q_wire.hxx"
#include "kernel/sg_husk/query/q_vert.hxx"
#include "kernel/kernapi/api/api.err"
#include "baseutil/debug/module.hxx"
```

Description:

This comprehensive API behaves analogous to the collection of like API's that return the related entities of a specific entity, such as api_get_faces and api_get_edges. This API however allows the specification of multiple entity types to be returned from a single pass of the traversal algorithm. The entity selection is made by passing a bit mask of topological ids, and optionally geometrical ids, to the API. The bit masks are created by 'or–ing' the respective ids together, as the following example demonstrates:

```
ENTITY_ID topo_bits = FACE_ID | EDGE_ID;
ENTITY_ID geom._bits = SURFACE_ID | CURVE_ID;
```

The entity id bit masks are categorized into topology ids and geometry ids, and cannot be mixed. They are however, mutually exclusive within their respective groups.

The following topological ids are available:

BODY_ID, LUMP_ID, SHELL_ID, SUBSHELL_ID, WIRE_ID, FACE_ID, LOOP_ID, COEDGE_ID, EDGE_ID, VERTEX_ID

The following geometrical ids are available:

TRANSFORM_ID, APOINT_ID, PCURVE_ID, SURFACE_ID, CURVE_ID

Given a set of topological id selections, the traversal algorithm searches for the selected entities from the level of the input entity within the topological hierarchy, and works its way down, selecting all that are lower in the hierarchical order. When the selection set contains entity ids that are higher in the topological hierarchy than the input entity, then the higher–level entities that share the input entities are also selected.

Given a FACE input entity with LUMP_ID and LOOP_ID selection ids, for example, the algorithm would select the owning LUMP, ignoring others, and would halt the traversal after selecting all loops of the face since the LOOP_ID is the lowest selection id.

The algorithm does not traverse laterally and will simply select the input entity in this case. For example, given a FACE input entity and a FACE_ID selection, the input face would be returned.

The geometrical id selections drive the algorithm in the same manner and assume the same level in the hierarchy as their topological owners.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – (default) patterned objects are created if they do not already exist, and are included in the list.

PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.

PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_entity_box

Function: Model Topology

Action: Gets the bounding box for a list of entities relative to the active working coordinate system.

Prototype:
```
outcome api_get_entity_box (
    const ENTITY_LIST& ent_list,// list of entities
    WCS* wcs,                   // WCS to use or NULL
                                // (model space)
    SPAposition& min_pt,        // minimum position
                                // of bounding box
                                // returned
    SPAposition& max_pt,        // maximum position
                                // of bounding box
                                // returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/geomhusk/wcs.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: If wcs is specified, then the bounding box computes relative to that WCS. The positions (min_pt and max_pt) that are returned as the corners of the bounding box are always returned relative to model space. Use care in interpreting the results. Consider the following code example:

This function is not guaranteed to return the tightest bounding box on spheres and tori. There are two options tight_sphere_box and tight_torus_box that must be set to get this.

```
outcome result;
BODY* box;
WCS* wcs1;
ENTITY_LIST elist;
SPAposition pt1(0,0,0);
SPAposition pt2(1,2,3);
SPAposition xpt(–1,0,0);
SPAposition ypt(0,–1,0);
```

```
result = api_solid_block(pt1, pt2, box);
    elist.add(box);
result = api_wcs_create(pt1, xpt, ypt, wcs1);

SPAposition min_pt, max_pt;
result = api_get_entity_box(elist, wcs1, min_pt,
                            max_pt);
```

This code example creates a box with corners at (0, 0, 0) and (1, 2, 3), and a WCS that is model space rotated about the *z*-axis by 180 degrees.

Relative to wcs1, the original corners of the box are (0, 0, 0) and (–1, –2, 3). The extrema relative to wcs are (–1, –2, 0) and (0, 0, 3). When these results are mapped back to model space, they are (1, 2, 0) and (0, 0, 3).

Because the API computes the extrema relative to wcs1 and returns the results mapped back into model space, the returned positions are:

min_pt	= (1,2,0)
max_pt	= (0,0,3)

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_entity_from_id

Function: History and Roll

Action: Returns an ENTITY identified by the given id.

Prototype:
```
outcome api_get_entity_from_id (
    tag_id_type id,          // id of the ENTITY
    ENTITY*& returned_ent,   // returned ENTITY
    HISTORY_STREAM* hs       // history where the
        = NULL               // ENTITY lives
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kerndata/bulletin/bulletin.hxx”
#include ”kernel/kerndata/data/container.hxx”
```

Description: Returns the pointer to the ENTITY identified by id. If no HISTORY_STREAM is specified, the default stream is used. If the ENTITY corresponding to the id is not alive, a NULL pointer is returned.

Errors: id is not valid in the given stream.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_entity_id

Function: History and Roll

Action: Returns a unique integer identifier for a given ENTITY.

Prototype:
```
outcome api_get_entity_id (
    ENTITY* ent,            // ENTITY for which id
                            // tag is requested
    tag_id_type& id         // returned id
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kerndata/data/container.hxx"
```

Description: Returns a unique integer, in a particular HISTORY_STREAM, for a given ENTITY. This id number (tag) is evaluated lazily but, once requested, is saved with the HISTORY_STREAM and does not change on restore.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_faces

Function: Model Topology

Action: Gets all faces related to an entity.

Prototype:
```
outcome api_get_faces (
    ENTITY* ent,             // entity to examine
    ENTITY_LIST& face_list,  // faces related to
                             // entity
    PAT_NEXT_TYPE include_pat// how to treat
        = PAT_CAN_CREATE,    // patterned faces
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernutil/law/pattern_enum.hxx"
```

Description: If the input entity (ent) is a BODY, LUMP, or SHELL, this API returns all FACEs of that entity. If the input entity is an EDGE, LOOP, or VERTEX, this API returns all FACEs that share the EDGE, LOOP, or VERTEX.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – patterned objects are created if they do not already exist, and are included in the list.

PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.

PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_file_info

Function: SAT Save and Restore

Action: Gets header info from the last restored file.

Prototype:

```
outcome api_get_file_info (
    FileInfo& info          // file information
                            // returned
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/savres/fileinfo.hxx"
```

Description: The API fills in a FileInfo class with the header information from the last restored file. It does not alter the model.

The FileInfo class contains the following information:

product_id is a string indicating the product and version which produced the save file.

date is a string indicating the date the model was saved (e.g., “Fri Feb 9 16:49:43 MST 1996”).

units is a double indicating the modeling units.

acis_version is a string indicating the version of the ACIS libraries used in the product which produced the save file.

file_version is the ACIS save file version for which the model was saved (e.g., 200).

SPAresabs is the distance tolerance in effect when the model was saved.

SPAresnor is the normal tolerance in effect when the model was saved.

For consistency, the recommended values for units are:

“mm” ..... = Millimeters
“cm” ..... = Centimeters
“M” ..... = Meters
“KM” ..... = Kilometers
“um” ..... = Microns
“In” ..... = Inches
“M” ..... = Meters
“Ft” ..... = Feet
“Mi” ..... = Miles
“mil” ..... = Mils

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_history_from_entity

Function: History and Roll

Action: Returns the HISTORY_STREAM in which the ENTITY lives.

Prototype:
```
outcome api_get_history_from_entity (
    ENTITY* ent,              // input entity
    HISTORY_STREAM*& hs       // returned history
                              // stream
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/kerndata/data/entity.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: Logging must be used.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_history_size

Function: History and Roll

Action: Gets the size of the DELTA_STATE in the HISTORY_STREAM.

Prototype:
```
outcome api_get_history_size (
    HISTORY_STREAM* hs,       // stream to use
    int& size,                // size
    DELTA_STATE* start_ds     // start
        = NULL
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: Refer to Action.

Errors: The pointer to the HISTORY_STREAM is NULL.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_journal

Function: ACIS Journal

Action: Gets a reference to the AcisJournal contained in AcisOptions.

Prototype:
```
outcome api_get_journal (
    AcisOptions* ao,         // acis options
    AcisJournal*& aj         // output reference to
                             // acis journal
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/acis_journal.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernapi/api/api.hxx"
```

Description: Gets a reference to the AcisJournal object contained in the AcisOptionsInternal data member.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/acis_journal.hxx

Effect: System routine

api_get_loops

Function: Model Topology

Action: Gets all loops related to an entity.

Prototype:
```
outcome api_get_loops (
    ENTITY* ent,             // entity to examine
    ENTITY_LIST& loop_list,  // loops related to
                             // entity returned
    PAT_NEXT_TYPE include_pat// how to treat
        = PAT_CAN_CREATE,    // patterned loops
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernutil/law/pattern_enum.hxx"
```

Description: If the input entity (ent) is a BODY, LUMP, FACE, EDGE, SHELL, or VERTEX, this API returns all LOOPs of that entity.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – patterned objects are created if they do not already exist, and are included in the list.

PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.

PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_lumps

Function: Model Topology

Action: Gets all lumps related to an entity.

Prototype:

```
outcome api_get_lumps (
    ENTITY* ent,             // entity to examine
    ENTITY_LIST& lump_list,  // lumps related to
                             // entity returned
    PAT_NEXT_TYPE include_pat// how to treat
        = PAT_CAN_CREATE,    // patterned lumps
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernutil/law/pattern_enum.hxx"
```

Description: If the input entity (ent) is a BODY, SHELL, FACE, EDGE, LOOP, or VERTEX, this API returns all LUMPs of that entity.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – patterned objects are created if they do not already exist, and are included in the list.

PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.

PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_modified_faces

Function: History and Roll

Action: Finds faces that have been deleted, created, or modified since a particular state.

Prototype:
```
outcome api_get_modified_faces (
    DELTA_STATE* ds,            // start state
    ENTITY_LIST& deleted_faces, // deleted since
                                // start
    ENTITY_LIST& created_faces, // created since
                                // start
    ENTITY_LIST& modified_faces,// modified since
                                // start
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/bulletin/bulletin.hxx″
#include ″kernel/kerndata/lists/lists.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This routine is intended to find lists of faces that have been created, deleted, or modified between the (input) start state and the current state of the history stream that contains that state.

For the purposes of this API, a face is not considered modified if its associated attributes or bounding box changes, but it is considered modified if one of its ”contained” entities is modified. These contained entities are its surface, loops, coedges, edges (and associated curves) and vertices (and associationed points).

The intended use of this API is to allow customers to avoid refaceting faces which can be determined (by examining the history stream) to be unchanged since the start state. Because of this, the algorithm to identify ”modified” faces is conservative: whenever it is unclear whether a change recorded in the history stream actually affected a face in a manner which requires refaceting, that face is included in the ”modified” list. This ensures that all faces which require refaceting will be included at one of the lists, at the expense of introducing occasional ”false positives” into the ”modified” list.

This API clears the deleted, created, and modified lists before writing to them (it overwrites them).

Errors: Roll back state not on history stream main branch.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_owner

Function: Model Topology

Action: Gets the top level owner of an entity.

Prototype:
```
outcome api_get_owner (
    ENTITY* ent,              // entity to determine
                              // owner
    ENTITY*& owner,           // top level owner of
                              // entity returned
    AcisOptions* ao = NULL    // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API finds the top level entity that ‘‘owns’’ the given entity (ent). For the purpose of this API, an entity owns another entity if the second entity is part of the definition of the first.

For example, if an edge is created with one of the curve-creation API functions, that edge is not owned by any other entity. If a solid is created, the edges of that solid are owned by the solid. For an EDGE, FACE, VERTEX, etc., that is a part of a solid, this API returns the BODY pointer of the solid. If the entity is not owned by another entity, then the pointer returns itself.

An ENTITY is top level when making a call to api_get_owner returns itself. Also, every ENTITY contains an owner method. This method would return the next higher ENTITY. If that object is the top level ENTITY, then this pointer is returned. This means that if a FACE does not point to an owning SHELL, this FACE is top level for that model. A BODY is normally top level, but in some cases, there are others that are the top level ENTITY.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_save_version

Function: SAT Save and Restore

Action: Gets the current save file format version.

Prototype:
```
outcome api_get_save_version (
    int& major_version,      // major version returned
                             // e.g., 1
    int& minor_version       // minor version returned
                             // e.g., 5
    );
```

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

Description: This API gets the output file format.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_shells

Function: Model Topology

Action: Gets all shells related to an entity.

Prototype:
```
outcome api_get_shells (
    ENTITY* ent,             // entity to examine
    ENTITY_LIST& shell_list, // shells related to
                             // entity returned
    PAT_NEXT_TYPE include_pat// how to treat
        = PAT_CAN_CREATE,    // patterned shells
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/data/entity.hxx″
#include ″kernel/kerndata/lists/lists.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
#include ″kernel/kernutil/law/pattern_enum.hxx″
```

Description: If the input entity (ent) is a BODY, LUMP, FACE, EDGE, LOOP, or VERTEX, this API returns all SHELLs of that entity.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – patterned objects are created if they do not already exist, and are included in the list.

PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.

PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_state_from_id

Function: History and Roll

Action: Returns a DELTA_STATE identified by the given id.

Prototype:
```
outcome api_get_state_from_id (
    STATE_ID id,             // id of the DELTA_STATE
    DELTA_STATE*& returned_ds,// returned DELTA_STATE
    HISTORY_STREAM* hs       // history where the
        = NULL               // ENTITY lives
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/bulletin/bulletin.hxx″
```

Description: Returns the pointer to the DELTA_STATE identified by id. If no HISTORY_STREAM is specified, the default stream is used. If the DELTA_STATE corresponding to the id is not in the stream, a NULL pointer is returned.

Errors: id is not valid in the given stream.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_state_id

Function: History and Roll

Action: Returns a unique integer identifier for a given DELTA_STATE.

Prototype:
```
outcome api_get_state_id (
    DELTA_STATE* ds,         // DELTA_STATE for which
                             // id tag is requested
    STATE_ID& id             // returned id
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/bulletin/bulletin.hxx”
```

Description: Returns a unique integer, in a particular HISTORY_STREAM, for a given DELTA_STATE. This id number (tag) is saved with the HISTORY_STREAM and does not change on restore.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_tcoedges

Function: Model Topology, Tolerant Modeling

Action: Gets all the tcoedges related to an entity.

Prototype:
```
outcome api_get_tcoedges (
    ENTITY* ent,            // entity to examine
    ENTITY_LIST& tcoedge_list   // tcoedges related
                            // to entity returned
    PAT_NEXT_TYPE include_pat// how to treat
        = PAT_CAN_CREATE,   // patterned tcoedges
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernutil/law/pattern_enum.hxx"
```

Description: If the input entity has COEDGEs; i.e., BODY, LUMP, FACE, etc., this function returns all COEDGEs of the entity.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – patterned objects are created if they do not already exist, and are included in the list.

PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.

PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_tedges

Function: Model Topology, Tolerant Modeling

Action: Gets all the tedges related to an entity.

Prototype:
```
outcome api_get_tedges (
    ENTITY* ent,            // entity to examine
    ENTITY_LIST& tedge_list,// edges related to
                            // entity returned
    PAT_NEXT_TYPE include_pat// how to treat
        = PAT_CAN_CREATE,   // patterned tedges
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/data/entity.hxx″
#include ″kernel/kerndata/lists/lists.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
#include ″kernel/kernutil/law/pattern_enum.hxx″
```

Description: If the input entity has EDGEs; i.e., BODY, LUMP, FACE, etc., this function returns all EDGEs of the entity. The input entity can also be a VERTEX, in which case this function returns all EDGEs that share the common VERTEX.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – patterned objects are created if they do not already exist, and are included in the list.

PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.

PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_tvertices

Function: Model Topology, Tolerant Modeling

Action: Gets all TVERTEXes related to an entity.

Prototype:

```
outcome api_get_tvertices (
    ENTITY* ent,              // entity to examine
    ENTITY_LIST& tvertex_list,// vertices related
                              // to entity returned
    PAT_NEXT_TYPE include_pat// how to treat
        = PAT_CAN_CREATE,     // patterned
                              // TVERTEXes
    AcisOptions* ao = NULL    // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernutil/law/pattern_enum.hxx"
```

Description: If the specified entity is a BODY, LUMP, SHELL, FACE, EDGE, or LOOP, this function returns all TVERTEXes of that entity.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – patterned objects are created if they do not already exist, and are included in the list.

PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.

PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_version_tag

Function: History and Roll

Action: Gets the version tag from an ACISversion.

Prototype:

```
outcome api_get_version_tag(
    AcisVersion* av,        // ACIS version object
    int& tag                // tag of ACIS
                            // version object
    );

outcome api_get_version_tag(
    int major,              // major version number
    int minor,              // minor version number
    int point,              // point version number
    int& tag                // tag of this version
    );

outcome api_get_version_tag(
    int& tag                // tag of current ACIS
                            // executable
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "baseutil/version/vers.hxx"
```

Description: Returns the requested version tag.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_get_vertices

Function: Model Topology

Action: Gets all vertices related to an entity.

Prototype:

```
outcome api_get_vertices (
    ENTITY* ent,            // entity to examine
    ENTITY_LIST& vertex_list,// vertices related
                            // to entity returned
    PAT_NEXT_TYPE include_pat// how to treat
        = PAT_CAN_CREATE,   // patterned vertices
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernutil/law/pattern_enum.hxx"
```

Description: If the specified entity (ent) is a BODY, LUMP, SHELL, FACE, EDGE, or LOOP, this API returns all VERTEXes of that entity.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – patterned objects are created if they do not already exist, and are included in the list.

PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.

PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_get_wires

Function: Model Topology

Action: Gets all the wires related to an entity.

Prototype:
```
outcome api_get_wires (
    ENTITY* ent,             // entity to examine
    ENTITY_LIST& out_list,   // wires related to
                             // entity returned
    PAT_NEXT_TYPE include_pat// how to treat
        = PAT_CAN_CREATE,    // patterned wires
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernutil/law/pattern_enum.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: If the input entity ent has WIREs; i.e., BODY, LUMP, etc., this API returns all the WIREs of the entity.

By default, patterned objects are included in the list of entities. In general, however, the parameter include_pat determines how this function deals with such objects. The user may specify any one of the following through this argument:

PAT_CAN_CREATE – patterned objects are created if they do not already exist, and are included in the list.

PAT_NO_CREATE – only those patterned objects that have already been created are included in the list.

PAT_IGNORE – no patterned objects besides seed pattern objects are included in the list.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_hedgehog

Function: Viewing

Action: Creates a DL_item list of hairs to show a vector field.

Prototype:
```
outcome api_hedgehog (
    law* field,                 // vector field
    law* base,                  // base of field
    double* starts,             // min value in each
                                // dimension
    double* ends,               // max value in each
                                // dimension
    int dim,                    // size of starts
                                // and ends
    int* hairs,                 // number of hairs in
                                // each dimension
    ENTITY_LIST& return_item,   // list of hairs
                                // returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "lawutil/law_base.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernutil/law/hog_api.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: field is a law specifying the vectors to show (the hairs). base is a law specifying where the roots of the hairs lie.

dim specifies whether a one–dimensional, two–dimensional, or three–dimensional array of hairs is produced.

starts and ends are arrays of one, two, or three start points and end points, depending on dim.

hairs is an array containing one, two, or three values, depending on dim, specifying how many hairs are to be created between the start and end points.

return_item contains the list of hairs for display.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/hog_api.hxx

Effect: Read-only

api_hex_cylindrical_pattern

Function: Patterns

Action: Creates a hexagonal pattern with cylindrical symmetry.

Prototype:

```
outcome api_hex_cylindrical_pattern (
    pattern*& pat,           // created pattern
    const FACE* in_face,     // face defining
                             // the pattern
    int num_angular,         // # of pattern elements
                             // about cylinder axis
    int num_axial            // # of pattern elements
        = 1,                 // along cylinder axis
    double spacing           // spacing of pattern
        = 0.0,               // elements
    AcisOptions* ao = NULL   // acis options
    );

outcome api_hex_cylindrical_pattern (
    pattern*& pat,           // created pattern
    const SPAposition& center,// start position
    const SPAvector& normal, // direction of
                             // cylinder axis
    int num_angular,         // # of pattern elements
                             // about cylinder axis
    int num_axial            // # of pattern elements
        = 1,                 // along cylinder axis
    double spacing           // spacing of pattern
        = 0.0,               // elements
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/vector.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kernutil/law/pattern.hxx”
#include ”kernel/kernutil/law/pattern_api.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Creates a two–dimensional hexagonal pattern with cylindrical symmetry, with a radius and axis defined either by the center position and normal vector or by the cylindrical face in_face. The number of angular and axial elements in the pattern are set by num_angular and num_axial, respectively, and the distance between pattern elements by spacing. The pattern coordinates are specified in the order (angular, axial).

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAposition center(5, 0, 0);
SPAvector normal(0, 1, 0);
int num_angular = 8;
int num_axial = 6;double spacing = 3.0;
check_outcome(result =
api_hex_cylindrical_pattern(pat, center,  normal,
num_angular, num_axial, spacing));

// Create a cylinder
BODY* cylinder = NULL;
SPAposition bottom(0, 0, 0);
SPAposition top(0.5, 0, 0);
double maj_rad = 1.0;
double min_rad = 1.0;
check_outcome(result =
api_solid_cylinder_cone(bottom, top, maj_rad,
min_rad, maj_rad, NULL, cylinder));

// Apply the pattern to the cylinder
check_outcome(result =
api_set_entity_pattern(cylinder, pat));

// Clean up
pat–>remove();
```

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_hex_pattern

Function: Patterns

Action: Creates a hexagonal pattern in two or three dimensions.

Prototype:
```
outcome api_hex_pattern (
    pattern*& pat,          // created pattern
    const SPAvector& normal,// normal to pattern
    const SPAvector& x_vec, // starting axis
    int num_x,              // repeat in x
    int num_y,              // repeat in y
    int num_z               // # times to
        = 1,                // repeat in z
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/vector/vector.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernutil/law/pattern.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Creates a hexagonal pattern in two or three dimensions. For two–dimensional patterns, the normal parameter specifies the direction normal to the pattern plane; for three–dimensional patterns, it sets the z–direction. The x_vec argument defines the pattern's starting axis and displacement; num_x, num_y, and num_z set the number of repetitions in each dimension over which the pattern extends.

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAvector normal(0, 0, 1);
SPAvector x_vec(2, 0, 0);
int num_x = 4;
int num_y = 4;
int num_z = 4;
check_outcome(result = api_hex_pattern(pat, normal,
x_vec,  num_x, num_y, num_z));
```

```
// Create a sphere
BODY* sph = NULL;
SPAposition center(1, 1, 0);
double radius = 1.0;
check_outcome(result = api_solid_sphere(center,
radius, sph));

// Apply the pattern to the sphere
check_outcome(result = api_set_entity_pattern(sph,
pat));

// Clean up
pat->remove();
```

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_hook_annotations

Function: Feature Naming

Action: Traverses the active list of annotations and adds ATTRIB_ANNOTATIONs to hook them to the annotated entities.

Prototype:
```
outcome api_hook_annotations (
    is_fun is_function      // type of annotation
        = is_ANNOTATION,
    BULLETIN_BOARD* bb      // obsolete, ignored
        = NULL,
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: The flag for is_fun defaults to is_ANNOTATION. However, any is function for a class can be used. So, for example, to get the top vertex annotations from a sweep operation, this function can be passed is_SWEEP_ANNO_VERTEX_TOP as an argument.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_initialize_kernel

Function: Modeler Control, Entity, Model Geometry, Model Topology, Construction Geometry

Action: Initializes the kernel library.

Prototype:
```
outcome api_initialize_kernel ();
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_initialize_spline

Function: Modeler Control, Spline Interface

Action: Initializes the spline library.

Prototype:
```
outcome api_initialize_spline ();
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/spline/api/spl_api.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/spline/api/spl_api.hxx

Effect: System routine

api_integrate_law

Function: Laws, Geometric Analysis, Analyzing Models

Action: Integrates a law over a given domain to a given tolerance.

Prototype:
```
outcome api_integrate_law (
    law* input_law,          // law to be integrated
    double start,            // start of the domain
    double end,              // end of the domain
    double& answer,          // value of integration
    double tolerance         // optional tolerance for
        = 1E-12,             // the answer
    int min_level            // optional minimum
        = 2,                 // Romberg Table rows
    int* used_level          // optional number of
        = NULL               // Romberg rows returned
    );
```

Includes:
```
#include “kernel/acis.hxx”
#include “kernel/kernapi/api/api.hxx”
#include “kernel/kernapi/api/kernapi.hxx”
#include “lawutil/law_base.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_integrate_law_wrt

Function: Laws, Geometric Analysis, Analyzing Models

Action: Integrates a law over a given domain to a given tolerance with respect to a given variable.

Prototype:

```
outcome api_integrate_law_wrt (
    law* input_law,          // law to be integrated
    double start,            // start of the domain
    double end,              // end of the domain
    int wrt,                 // variable to integrate
                             // with respect to
    double* along,           // an array the size of
                             // the take dim of the
                             // law with all other
                             // variables filled in
    double& answer,          // value of integration
    double tolerance         // optional tolerance for
        = 1E-12,             // the answer
    int min_level            // optional minimum
        = 2,                 // Romberg Table rows
    int* used_level          // optional number of
        = NULL               // Romberg rows
                             // returned
    );

outcome api_integrate_law_wrt(
    law* integrand,          // law to be integrated
    double start,            // start of the domain
    double end,              // end of the domain
    int wrt,                 // variable to integrate
                             // with respect to
    double* along,           // an array the size of
                             // the take dim of the
                             // law with all other
                             // variables filled in
    double* answer,          // value of integration
    double tol,              // optional tolerance for
                             // the answer (default
                             // = 1.0E-12)
    int min_level,           // optional minimum
                             // Romberg Table rows
                             // (default = 2)
    int* used_level          // optional number of
                             // Romberg rows used
                             // (default = NULL)
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "lawutil/law_base.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_integrate_law_wrt_and_splits

Function: Laws, Geometric Analysis, Analyzing Models

Action: Integrates a law over a given domain to a given tolerance with respect to a given variable with respect an array of points to split the domain.

Prototype:

```
outcome api_integrate_law_wrt_and_splits (
    law* input_law,          // law to be integrated
    double start,            // start of the domain
    double end,              // end of the domain
    int wrt,                 // variable to integrate
                             // with respect to
    double* along,           // an array the size of
                             // the take dim of the
                             // law with all other
                             // variables filled in
    double& answer,          // value of integration
    int number_of_splits     // optional number of
        = 0,                 // singularities
    double* splits           // optional number of
        = NULL,              // splits
    double tolerance         // optional tolerance for
        = 1E-12,             // the answer
    int min_level            // optional minimum
        = 2,                 // Romberg Table rows
    int* used_level          // optional number of
        = NULL               // Romberg rows returned
    );
```

```
outcome  api_integrate_law_wrt_and_splits(
    law* integrand,          // law to be integrated
    double start,            // start of the domain
    double end,              // end of the domain
    int wrt,                 // variable to integrate
                             // with respect to
    double* along,           // an array the size of
                             // the take dim of the
                             // law with all other
                             // variables filled in
    double* answer,          // value of integration
    int number_of_splits,    // optional number of
                             // singularities
                             // (default = 0)
    double* splits,          // optional where the
                             // singularities are
                             // (default = NULL)
    double tol,              // optional tolerance for
                             // the answer (default
                             // = 1.0E-12)
    int min_level,           // optional minimum
                             // Romberg Table rows
                             // (default = 2)
    int* used_level          // optional number of
                             // Romberg rows used
                             // (default = NULL)
    );
```

Includes:

```
#include “kernel/acis.hxx”
#include “kernel/kernapi/api/api.hxx”
#include “kernel/kernapi/api/kernapi.hxx”
#include “lawutil/law_base.hxx”
```

Description: During the integration it will take into account an array of points to split the domain up into. This should be used if the domain contains known singularities.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_law_to_entity

Function: Laws

Action: Converts a law mathematic function into an entity for the purposes of saving to and restoring from a SAT file.

Prototype:
```
outcome api_law_to_entity (
    law* input_law,          // law function
    ENTITY*& out_ent,        // pointer to entity
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include “kernel/acis.hxx”
#include “kernel/kernapi/api/api.hxx”
#include “kernel/kernapi/api/kernapi.hxx”
#include “kernel/kerndata/data/entity.hxx”
#include “lawutil/law_base.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Law mathematic functions that are used for analysis of the design are not normally saved to the SAT file. Typically, only laws that are attached to model entities through geometry definitions are saved to the SAT file. In order to make laws more persistent and to share them from session to session, they can be turned into LAW instances, which are derived from ENTITY and are saved and restored.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_linear_pattern

Function: Patterns

Action: Creates a linear pattern.

Prototype:
```
outcome api_linear_pattern (
    pattern*& pat,           // created pattern
    const SPAvector& x_vec,  // displacement vector
                             // in the x-direction
    int num_x,               // # of elements in
                             // the x-direction
    const SPAvector& y_vec   // displacement vector
        =*(SPAvector*)NULL_REF,// in the y-direction
    int num_y                // # of elements in
        = 1,                 // the y-direction
    const SPAvector& z_vec   // displacement vector
        =*(SPAvector*)NULL_REF,// in the z-direction
    int num_z                // # of elements in
        = 1,                 // the z-direction
    logical y_staggered      // flag to stagger the
        = FALSE,             // pattern y-components
    logical z_staggered      // flag to stagger the
        = FALSE,             // pattern z-components
    logical fit_distance     // displacement flag
        = FALSE,
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "baseutil/vector/vector.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernutil/law/pattern.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Creates a linear pattern in one, two, or three dimensions, depending upon the number of input arguments. The pattern orientation is specified by x_vec, y_vec, and z_vec, which are neither required to be in the coordinate directions nor to be orthogonal. The number of repetitions along each axis is defined by num_x, num_y, and num_z If y_staggered or z_staggered is TRUE, the pattern is staggered along the associated directions. If fit_distance is TRUE, the vectors x_vec, y_vec, and z_vec represent displacements over the entire pattern rather than displacements between adjacent pattern elements.

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAvector x_vec(2, 0, 0);
int num_x = 4;
SPAvector y_vec(0, 2, 0);
int num_y = 3;
SPAvector z_vec(1, 1, 2);
int num_z = 3;
check_outcome(result = api_linear_pattern(pat, x_vec,
num_x, y_vec, num_y, z_vec, num_z));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
pat));

// Clean up
pat->remove();
```

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_linear_scale_pattern

Function: Patterns

Action: Creates a new pattern by applying a linear scale to an existing pattern.

Prototype:

```
outcome api_linear_scale_pattern (
    pattern*& pat,              // created pattern
    const pattern& in_pattern,  // input pattern
    double first_scale,         // first scale
    double last_scale,          // second scale
    int which_dim,              // dimension for
                                // scaling
    const SPAposition& root,    // position for
                                // scaling
    logical merge               // merge flag
        = TRUE,
    AcisOptions* ao = NULL      // acis options
    );

outcome api_linear_scale_pattern(
    pattern*& pat,              // created pattern
    const pattern& in_pattern,  // input pattern
    const SPAvector& first_scale,// first scale
    const SPAvector& last_scale,// second scale
    int which_dim,              // dimension for
                                // scaling
    const SPAposition& root,    // position for
                                // scaling
    logical merge = TRUE,       // merge flag
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernutil/law/pattern.hxx”
#include ”kernel/kernutil/law/pattern_api.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
#include ”baseutil/vector/vector.hxx”
```

Description: Applies a linear scale, from first_scale to last_scale (which may be given as vectors when nonuniform scaling is desired), to an existing pattern, merging with any existing scaling or, optionally (with merge=FALSE), replacing it. The argument which_dim specifies the dimension in which the scale is applied. The position root specifies the neutral point about which the scaling takes place (i.e., the point on the seed entity that remains fixed while the entity's dimensions are altered). Both first_scale and last_scale must be greater than zero.

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAvector x_vec(4.0, 0, 0);
int num_x = 8;
SPAvector y_vec(0, 2.0, 0);
int num_y = 10;
check_outcome(result = api_linear_pattern(pat, x_vec,
num_x,  y_vec, num_y));

// Modify the pattern
pattern* mod_pat = NULL;
double begin_scale = 0.5;
double end_scale = 2.0;
int which_dim = 0;
SPAposition root(0, 0, 0);
check_outcome(result =
api_linear_scale_pattern(mod_pat,  *pat,first_scale,
last_scale, which_dim, root));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad,  num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
mod_pat));

// Clean up
pat->remove();mod_pat->remove();
```

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_load_state

Function: SAT Save and Restore

Action: Loads the state of global variables from a given text file.

Prototype:
```
outcome api_load_state (
    FILE* file_ptr,          // file descriptor
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: This API loads the states of global variables such as options and tolerances from a given text file. You may use this function with api_save_state to compare the behaviors between your application and Scheme AIDE.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_logging

Function: History and Roll

Action: Sets logging on or off for roll back.

Prototype:
```
outcome api_logging (
    logical on_off           // TRUE for on
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″baseutil/logical.h″
```

Description: TRUE enables logging of data structure changes.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_loop_type

Function: Debugging, Model Topology

Action: Determines the type of a given loop.

Prototype:
```
outcome api_loop_type (
    LOOP* in_loop,          // loop to test
    loop_type& type,        // type of loop
    int info[]              // array holding
        = NULL              // information about loop
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/ptinface/ptfcenum.hxx″
#include ″kernel/kerndata/top/loop.hxx″
```

Description: This API returns the type of a given loop, types include:

- loop_unknown
- loop_periphery
- loop_hole
- loop_separation (not used)
- loop_u_separation
- loop_v_separation
- loop_uv_separation

They are enum types defined in ptfcenum.hxx

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_make_cubic

Function: Laws, Mathematics

Action: Creates a cubic law given {a,b,f(a),f(b),f'(a),f'(b)}.

Prototype:
```
outcome api_make_cubic (
    double aval,            // a value
    double bval,            // b value
    double faval,           // f at a
    double fbval,           // f at b
    double ffaval,          // deriv of f at a
    double ffbval,          // deriv of f at b
    law*& answer            // ptr to law
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "lawutil/law_base.hxx"
```

Description: Produces a cubic polynomial with given boundary conditions for both it and its first derivative. The user supplies the boundary values a and b, the desired output of the law at a and b (e.g., f_a and f_b), and the desired output of the first derivative at a and b (e.g., df_a and df_b). The result is a cubic polynomial meeting these boundary conditions.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_make_linear

Function: Laws, Mathematics

Action: Creates a linear law given {a,b,f(a),f(b)}.

Prototype:
```
outcome api_make_linear (
    double aval,            // a value
    double bval,            // b value
    double faval,           // f at a
    double fbval,           // f at b
    law*& answer            // ptr to law
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "lawutil/law_base.hxx"
```

Description: Produces a linear polynomial with given boundary conditions for both its output. The user supplies the boundary values a and b and the desired output of the law at a and b (e.g., f_a and f_b). The result is a linear polynomial meeting these boundary conditions.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_make_polynomial_law

Function: Laws, Mathematics

Action: Creates a polynomial law.

Prototype:
```
outcome api_make_polynomial_law (
    double* coeff,          // array of coefficients
    int degree,             // maximum degree of
                            // polynomial
    law*& answer            // ptr to law
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "lawutil/law_base.hxx"
```

Description: Given an array of coefficients and the maximum degree for the polynomial, this creates a law that represents the associated polynomial.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_make_quintic

Function: Laws, Mathematics

Action: Creates a quintic law given {a,b,f(a),f(b),f'(a),f'(b) f''(a) f''(b)}.

Prototype:

```
outcome api_make_quintic (
    double aval,            // a value
    double bval,            // b value
    double faval,           // f at a
    double fbval,           // f at b
    double ffaval,          // 1st deriv of f at a
    double ffbval,          // 1st deriv of f at b
    double fffaval,         // 2nd deriv of f at a
    double fffbval,         // 2nd deriv of f at b
    law*& answer            // ptr to law
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "lawutil/law_base.hxx"
```

Description: Produces a quintic polynomial with given boundary conditions for it, its first derivative, and its second derivative. The user supplies the boundary values a and b, the desired output of the law at a and b (e.g., f_a and f_b), the desired output of the first derivative at a and b (e.g., df_a and df_b), and the desired output of the second derivative at a and b (e.g., ddf_a and ddf_b). The result is a quintic polynomial meeting these boundary conditions.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_make_rails

Function: Sweeping, Laws

Action: Creates the default rail laws for sweeping along a wire.

Prototype:
```
outcome api_make_rails (
    ENTITY* path,           // a WIRE or EDGE
    law**& rails,           // array of rail
                            // laws returned
    int& number_of_rails,   // number of rail
                            // laws returned
    law** axis              // optional axis
        = NULL,             // in an array
    FACE** faces            // optional faces
        = NULL,             // in an array
    law** user_rails        // optional user
        = NULL,             // defined rails
    law* twist_law          // optional twist
        = NULL,             // law
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include “lawutil/law_base.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This produces an array of rail laws that can be used by sweeping in the sweep options. A single rail law is produced if the path is a single edge or a wire with a single underlying edge. Otherwise, it creates multiple rail laws, one for each underlying edge in the path.

The only required argument is the path. If no other arguments are supplied, then the default rails are created. The default for the creation of rails is:

– If the path is planar, the rail law is the planar normal. A constant vector law is returned
– If the path is a helix, the rail law points towards the axis. The Frenet law is returned.
– If all edges in the wire are planar, then an array of rail laws is created, whereby each law in the array corresponds to an edge in the wire. The rail laws correspond to the planar normal of edges.
– If the wire has surfaces, then the surface normal laws are returned.
– If the path isn't one of the above cases, the rail uses minimum rotation.

If the input path is composed of multiple pieces, such as a wire with more than one underlying edge, then array arguments must supply the same number of elements as the number of path elements. They may pad their array with NULL arguments.

The axis argument is used for path segments that have an implied center axis. An example of this might be a helix, an expanding helix, or the coil of a telephone handset cable. The axis argument is the derivative of the implied center axis, which tells the implied axis direction. When the axis is supplied, then its cross product with the path is returned. The axis array can be padded with NULL for sections of the path that do not have an implied axis.

The face argument is used when a portion of the path segments borders a non-analytic face. The coedge of the wire provided as path must actually belong to the face entity supplied. The face must be non-analytic. The resulting rail is oriented to the face normal. The face array can be padded with NULL for sections of the path that do not have such a face.

The user-rails argument permits any default rail for a given section of the path to be overridden by the user-supplied law in the array. The user-rails array can be padded with NULL for sections of the path that are to use the default.

The twist argument works on the whole rail array. After the other rail parameters have been input and calculated, the law provided by twist operates on the whole set of rails. This takes in an angle of twist per distance along the path.

Errors: None

Limitations: When faces are supplied, the coedge of the wire must actually belong to the face. The face must be non-analytic. The face argument is not supported for analytic geometry in the face.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_make_root_state

Function: History and Roll

Action: Sets the state of the root.

Prototype:
```
outcome api_make_root_state (
    DELTA_STATE* ds          // state to make the root
                             // of its stream
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: For any given ENTITY, the history of that ENTITY must be maintained on a single HISTORY_STREAM. api_distribute_state_to_streams implements a consistency check to make sure this is the case. When processing a change or delete BULLETIN, it first determines which HISTORY_STREAM should get the BULLETIN. It then checks to see that the given stream contains a create BULLETIN for the ENTITY. This depends on cooperation from api_prune_history. api_prune_history may delete a number of DELTA_STATES, but it retains the create BULLETINs for any active ENTITYs in the root DELTA_STATE.

In addition to holding create BULLETINs from pruned DELTA_STATEs, the root DELTA_STATE cannot be rolled over by api_roll_n_states, api_change_to_state, or other high level roll APIs. (The low level api_change_state can roll over the root state, but we recommend against it.) This makes it useful for holding BULLETINs created during application initializations. For example, one does not typically want to be able to roll over the create BULLETINs from loading a SAT file into a new part.

api_make_root_state prunes away all previous history, saving the create BULLETINs at the beginning of the given state, and makes the state the root state.

Errors: The pointer to ds is NULL.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_make_VBL_output_surfaces

Function: Blending

Action: Splits the approximating surface for a VBL_SURF into n four–sided bs3_surface patches.

Prototype:
```
outcome api_make_VBL_output_surfaces (
    const surface* vbl_sf,   // surface with
                             // underlying VBL_SURF
    double& interior_fit,    // achieved interior fit
                             // tolerance
    double& boundary_fit,    // achieved boundary fit
                             // tolerance
    bs3_surface*& bs3_array,// array of returned
                             // bs3_surfaces
    int& n,                  // number of returned
                             // bs3_surfaces
     logical approx_error    // if TRUE, approximate
        = TRUE               // the error
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerngeom/surface/surdef.hxx"
#include "kernel/spline/bs3_srf/bs3surf.hxx"
#include "baseutil/logical.h"
```

Description: This API approximates an *n*–sided VBL_SURF by *n* four–sided bs3_surfaces, which enables ACIS to output vertex blend surfaces in a form that can be read and used by most other packages.

In ACIS, the VBL_SURF is approximated by a bs3_surface whose parameterization is defined over an *n*–sided regular polygon. This API makes up *n* four–sided patches that are separated from one another in the original approximating surface parameter space by straight lines radiating from the center of the *n*–sided polygon to the midpoint of each of the sides. The new approximating surfaces are parameterized over the unit square. Adjacent bs3_surface patches are made up to have the same number of knot points along common boundaries, and so the surfaces are C0 continuous across the boundaries. However, the parameter lines do not run smoothly across the boundaries, and so the approximation is not C1 or G1 continuous across the patch boundaries.

The API function receives a pointer to a surface, which has an underlying VBL_SURF. The caller should also supply two doubles, which specify the requested fit tolerances of the approximating surfaces, both on the interior of the VBL_SURF, and on its boundary. These will return the fit tolerances that the approximating surfaces have achieved, which may be larger than the requested fit tolerances, if these are particularly small, or if the surface is particularly complex. One might, for example, request an internal fit tolerance of .001 and a boundary fit tolerance that is 10 times smaller than this. Note that the interior fit tolerance (but NOT the boundary one) may be passed as exactly –1.0 to mean ”do not measure the interior fit”, in which case no particular interior fit is guaranteed, except that which comes about naturally by having fit the boundary correctly, and no meaningful value is returned for the achieved interior fit. The function can operate more quickly if no specific interior fit is requested. The internal joins between the patch boundaries are unaffected by this.

The caller must also supply the function with a pointer to a bs3_surface and a reference to an integer. These will be used to return an array of the approximating bs3_surfaces and the length of this array, respectively. Additionally, an approx_error flag can be supplied which specifies whether the errors returned need to be precise or merely a (close) upper bound. If passed TRUE, the error is bounded approximately but quite closely, and the function will be able to work more quickly.

If the supplied surface is of type spline and the underlying spl_sur is a VBL_SURF, this API function makes up *n* four–sided approximating bs3_surfaces. A pointer to an array of these bs3_surfaces is then returned, along with the number of bs3_surfaces in the array, and the maximum internal and boundary fit tolerances of the bs3_surfaces. If the supplied surface is not a spline, or if it is a spline and the underlying spl_sur is not a VBL_SURF, this API function returns a NULL bs3_surface pointer and sets the number of approximating bs3_surfaces to zero.

Errors: A NULL pointer to a surface is specified. The spl_sur underlying the surface is not a VBL_SURF.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_make_version_object

Function: History and Roll

Action: Makes an AcisVersion object from various forms of input.

Prototype:

```
outcome api_make_version_object(
    AcisVersion*& av,       // ACIS version object
                            // from following input
    int tag                 // input tag
    );

outcome api_make_version_object(
    AcisVersion*& av,       // ACIS version object
                            // from following input
    int major,              // major version number
    int minor,              // minor version number
    int point               // point version number
    );

outcome api_make_version_object(
    AcisVersion*& av        // ACIS version object
                            // of current executable
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "baseutil/version/vers.hxx"
```

Description: Makes an AcisVersion object from various forms of input.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_merge_states

Function: History and Roll

Action: Modifies a history stream by merging a range of delta states.

Prototype:

```
outcome api_merge_states (
    DELTA_STATE* ds1        // state defining one end
        = NULL,             // of range to be merged
    DELTA_STATE* ds2        // other end
        = NULL,             // of range to be merged
    HISTORY_STREAM* hs      // history stream
        = NULL,             // containing states
    logical prune_partners  // flag to allow pruning
        = FALSE,            // of partner states
    logical keep_both       // flag to indicate both
        = FALSE             // states are to be kept
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/bulletin/bulletin.hxx”
#include ”baseutil/logical.h”
```

Description: This API merges the delta states contained in a range specified by the user through the arguments ds1 and ds2. If one of these arguments is given as NULL, the specified state is merged with its next state. If both are given as NULL, the active delta state is merged with its predecessor. The user may specify the relevant history stream by furnishing the argument hs. Otherwise, it is taken from ds1 or ds2, if they are given, or set to the default stream, if they are not. By default, the function fails if the range contains states having partner states, but if the flag prune_partners is set to TRUE, the function will prune the branches associated with these partners. If the keep_both flag is TRUE, the merge happens between the given states so neither is deleted. The keep_both flag has no effect unless two non–NULL states are given.

Errors: The delta states referenced by ds1 and ds2 do not belong to the same stream, they do not belong to the specified stream, or partner states were encountered with prune_partners set to FALSE.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_name_state

Function: History and Roll

Action: Names the current state.

Prototype:
```
outcome api_name_state (
    const char* name,        // name to give to
                             // current operation
    DELTA_STATE* ds          // delta state to name
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/bulletin/bulletin.hxx″
```

Description: This API assigns a name to the given delta state. Use the specified name in calls to api_find_named_state to find a state to pass to api_change_to_state.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_ndifferentiate_law

Function: Laws, Geometric Analysis, Analyzing Models

Action: Numerically differentiates a law at a given point with respect to a given variable a given number of times.

Prototype:
```
outcome api_ndifferentiate_law (
    law* input_law,          // law to differentiate
    double* where,           // where to take the
                             // derivative
    int which_dim,           // which variable to take
                             // the derivative with
                             // respect to
    double* answer,          // 0 = normal, 1 = from
                             // the left, 2– from the
                             // right
    int type                 // how many times to take
        = 0,                 // the derivative
    int times                // value of
        = 1                  // differentiation
                             // returned
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include “lawutil/law_base.hxx”
```

Description: The derivative may be taken from both sides or just from the left or right.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_nmax_of_law

Function: Laws, Geometric Analysis, Analyzing Models

Action: Gets the maximum value of a given law over the given domain.

Prototype:
```
outcome api_nmax_of_law (
    law* input_law,          // law to find the
                             // roots of
    double start,            // start of the domain
    double end,              // end of the domain
    double* answer           // returns where the
                             // maximum value is
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include “lawutil/law_base.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_nmin_of_law

Function: Laws, Geometric Analysis, Analyzing Models

Action: Gets the minimum value of a given law over the given domain.

Prototype:
```
outcome api_nmin_of_law (
    law* input_law,          // law to find the roots
                             // of
    double start,            // start of the domain
    double end,              // end of the domain
    double* answer           // returns where the
                             // minimum value is
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include “lawutil/law_base.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_note_state

Function: History and Roll

Action: Sets a check point for roll back and returns model differences since previous call to api_note_state.

Prototype:
```
outcome api_note_state (
    DELTA_STATE*& ds,        // state change returned
    HISTORY_STREAM* hs       // history stream
        = NULL,              // to check
    logical delete_if_empty  // flag to delete
        = FALSE              // if empty
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "baseutil/logical.h"
```

Description: This API notes the current state of the model and returns a pointer to a delta_state that contains differential model data covering the period since the previous call to api_note_state.

If there have been no model changes since the last call to api_note_state the returned DELTA_STATE* will be NULL.

The default HISTORY_STREAM is used, unless a different history stream is supplied.

If the logical delete_if_empty is TRUE, an empty DELTA_STATE (i.e. one with no bulletins) will be removed from the stream when noted.

To return the model to the previous state from the current state, call api_change_state with the delta_state as argument.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_nroots_of_law

Function: Laws, Geometric Analysis, Analyzing Models

Action: Gets all the roots of a given law over the given domain.

Prototype:
```
outcome api_nroots_of_law (
    law* input_law,          // law to find the roots
                             // of
    double start,            // start of the domain
    double end,              // end of the domain
    int* size,               // how many roots where
                             // found returned
    double** answer          // returns where the
                             // maximum value is
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include “lawutil/law_base.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_nsolve_laws

Function: Laws, Geometric Analysis, Analyzing Models

Action: Determines where two given laws are equal over a given domain.

Prototype:
```
outcome api_nsolve_laws (
    law* input_law1,         // first law to solve
                             // with
    law* input_law2,         // second law to solve
                             // with
    double start,            // start of the domain
    double end,              // end of the domain
    int* size,               // returns how many
                             // places the laws equal
                             // each other
    double** answer          // returns where the two
                             // laws equal each other
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include “lawutil/law_base.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: The number of places that the two laws equal must be finite.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_optimize_tvertex_tolerance

Function: Precision and Tolerance, Tolerant Modeling

Action: Optimize (minimizes) the TVERTEX tolerance on the ends of a EDGE or TEDGE.

Prototype:
```
outcome api_optimize_tvertex_tolerance (
    EDGE* this_edge,        //edge to optimize
    AcisOptions* ao         //ACIS options such as
        = NULL              //version and journal
    );
```

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

Description: Optimize (minimizes) the TVERTEX tolerance on the ends of a EDGE or TEDGE

Errors: None

Limitations: Not applicable

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_pattern_find_bump

Function: Patterns

Action: Finds the bump associated with a given face or loop.

Prototype:
```
outcome api_pattern_find_bump (
    const ENTITY* seed,      // input face or lump
    ENTITY_LIST& face_list,  // faces belonging
                             // to bump
    ENTITY_LIST& loop_list,  // loops belonging
                             // to bump
    ENTITY_LIST& no_cross_list// faces limiting
        =*(ENTITY_LIST*)NULL_REF,    // bump's extent
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernutil/law/pattern_api.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Finds the bump associated with the face or loop specified by seed, and returns a list of its faces and loops in face_list and loop_list. The extent of the bump's definition may be limited by including a no_cross_list of faces.

Errors: The seed used to find the bump is neither a face nor a loop.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: System routine

api_pattern_to_entity

Function: Patterns, SAT Save and Restore

Action: Converts a pattern into an entity for the purposes of saving to and restoring from a SAT file.

Prototype:
```
outcome api_pattern_to_entity (
    pattern* in_pat,         // pattern to convert
    ENTITY*& out_ent,        // instance returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kernutil/law/pattern.hxx"
#include "kernel/kernutil/law/patternent.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: In order to make patterns more persistent and to share them from session to session, they can be turned into APATTERN instances, which are derived from ENTITY and are saved and restored.

Errors: The specified pattern is NULL.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/patternent.hxx

Effect: Read–only

api_pause_journal

Function: ACIS Journal

Action: Sets the status flag for journalizing to off, disabling the snapshot journal mechanism.

Prototype:
```
outcome api_pause_journal (
    AcisOptions* ao         // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/acis_journal.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernapi/api/api.hxx"
```

Description: Sets the status flag to off to disable journalizing.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/acis_journal.hxx

Effect: System routine

api_periodic_keep_pattern

Function: Patterns

Action: Creates a new pattern by applying a periodic keep–filter to an existing pattern.

Prototype:
```
outcome api_periodic_keep_pattern (
    pattern*& pat,          // created pattern
    const pattern& in_pattern,  // input pattern
    const logical* keep,    // array of keep values
    int period,             // # of keep values
    int which_dim,          // dimension for filter
    logical merge           // merge flag
        = TRUE,
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernutil/law/pattern.hxx"
#include "kernel/kernutil/law/pattern_api.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Applies a periodic keep–filter to an existing pattern, merging with any existing keep law or, optionally (with merge=FALSE), replacing it. The argument keep is the Boolean list of successive keep values, so that the size of the list (period) is the periodicity of the filter. The argument which_dim specifies the dimension within which the filter is applied.

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAvector x_vec(4.0, 0, 0);
int num_x = 6;
SPAvector y_vec(0, 2.0, 0);
int num_y = 12;
SPAvector z_vec(0, 0, 3.0);
int num_z = 4;
check_outcome(result = api_linear_pattern(pat, x_vec,
num_x, y_vec, num_y, z_vec, num_z));
```

```
// Modify the pattern
pattern* mod_pat = NULL;
logical keep[3];keep[0] = TRUE;
keep[1] = TRUE;
keep[2] = FALSE;
int which_dim = 1;
check_outcome(result =
api_periodic_keep_pattern(mod_pat, *pat,  keep, 3,
which_dim));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
mod_pat));

// Clean up
pat->remove();
mod_pat->remove();
```

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_periodic_scale_pattern

Function: Patterns

Action: Creates a new pattern by applying a periodic scale to an existing pattern.

Prototype:

```
outcome api_periodic_scale_pattern (
    pattern*& pat,           // created pattern
    const pattern& in_pattern,// input pattern
    const double* scale,     // array of scale values
    int period,              // # of scale values
    int which_dim,           // dimension for scaling
    const SPAposition& root,// base position
    logical merge            // merge flag
        = TRUE,
    AcisOptions* ao = NULL   // acis options
    );

outcome api_periodic_scale_pattern(
    pattern*& pat,           // created pattern
    const pattern& in_pattern,// input pattern
    const SPAvector* scale,  // array of scale values
    int period,              // number of scale values
    int which_dim,           // dimension for scaling
    const SPAposition& root,// base position
    logical merge = TRUE,    // merge flag
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernutil/law/pattern.hxx"
#include "kernel/kernutil/law/pattern_api.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "baseutil/vector/vector.hxx"
```

Description: Applies a periodic scale to an existing pattern, merging with any existing scaling or, optionally (with merge=FALSE), replacing it. The argument scale is the list of the successive scale values, and can be given as a vector list when nonuniform scaling is desired, so that the size of the list (period) is the periodicity of the scaling pattern. The argument which_dim specifies the dimension in which the scale is applied. The position root specifies the neutral point about which the scaling takes place (i.e., the point on the seed entity that remains fixed while the entity's dimensions are altered). All scale values in the list must be greater than zero.

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAvector x_vec(4.0, 0, 0);
int num_x = 8;
SPAvector y_vec(0, 2.0, 0);
int num_y = 10;
check_outcome(result = api_linear_pattern(pat, x_vec,
num_x,  y_vec, num_y));

// Modify the pattern
pattern* mod_pat = NULL;
double scale[4];
scale[0] = 0.5;
scale[1] = 1.5;
scale[2] = 1.0;
scale[3] = 2.0;
int which_dim = 0;
SPAposition root(0, 0, 0);
check_outcome(result =
api_periodic_scale_pattern(mod_pat,   *pat, scale,
4, which_dim, root));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
mod_pat));

// Clean up
pat->remove();
mod_pat->remove();
```

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_polar_grid_pattern

Function: Patterns

Action: Creates a polar–grid pattern.

Prototype:
```
outcome api_polar_grid_pattern (
    pattern*& pat,           // created pattern
    const SPAposition& center,// center (root)
                             // position
    const SPAvector& normal, // normal to pattern
                             // plane
    int num_rings,           // # of rings in pattern
    double distance,         // distance between
                             // pattern rings
    const SPAvector& start   // pattern start
        =*(SPAvector*)NULL_REF,// direction
    logical not_rotate       // rotation flag
        = FALSE,
    logical hex_symmetry     // force hex symmetry
        = FALSE,             // flag
    double start_angle       // start angle
        = 0.0,
    double end_angle         // end angle
        = 2.0* 3.14159265358979323846,
    double ratio             // ratio of minor/major
        = 1.0,               // radii
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/vector.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernutil/law/pattern.hxx”
#include ”kernel/kernutil/law/pattern_api.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Creates a two–dimensional polar–grid pattern defined by a root position center (which may or may not lie upon the seed entity) and the vector normal, which sets the orientation of the pattern. The number of rings in the grid (including the center) is specified by num_rings, and the distance between rings by spacing. The optional start argument specifies the direction of the first spoke of the pattern. The elements of the pattern are kept in a fixed orientation if not_rotate is TRUE; setting hex_symmetry to TRUE ensures that hexagonal symmetry is maintained for patterns extending either 360 or 180 degrees. The start_angle and end_angle arguments fix the angular extent of the pattern, in radians, and the ratio argument sets the ratio of minor/major radii of the pattern perimeter. The pattern coordinates are specified in the order (radial, angular).

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAposition center(0, 0, 0);
SPAvector normal(0, 0, 1);
int num_rings = 5;
double spacing = 4.0;
check_outcome(result = api_polar_grid_pattern(pat,
center,normal, num_rings, spacing));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
pat));

// Clean up
pat->remove();
```

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_project_curve_to_surface

Function: Model Geometry

Action: Projects a curve onto a surface.

Prototype:
```
outcome api_project_curve_to_surface (
    const curve& in_curve,      // curve to project
    const SPAinterval& in_range,// parameter range of
                                // in_curve
    const surface& in_surface,  // surface to project
                                // curve on to
    curve*& curve_on_surface,   // resulting curve
                                // projected onto
                                // surface
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/vector/interval.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerngeom/curve/curdef.hxx"
#include "kernel/kerngeom/surface/surdef.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API projects in_curve onto in_surface, returning the result in is_surface. in_range is the parameter range of in_curve.

The input curve and surface should both be in world coordinates. They might not be if their owning body contains a transformation, and may need to be converted. See the Scheme command, edge:project-to-face, for an example of how to do this.

Only that part of the curve for which a perpendicular projection onto the surface exists will be projected. Parts which can only be projected to the boundaries of the surface will be excluded.

Errors: Curve or surface not in world coordinates.

Limitations: If the curve has a perpendicular projection onto the surface over more than one distinct interval, the function will fail, as it can only return a single output curve.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_prune_following

Function: History and Roll

Action: Removes forward delta states from a history stream.

Prototype:
```
outcome api_prune_following (
    HISTORY_STREAM* hs      // history stream
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: Prunes away all branches of the history stream following the active state (see api_prune_history).

Errors: The pointer hs is NULL.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_prune_history

Function: History and Roll

Action: Removes delta states from a history stream.

Prototype:
```
outcome api_prune_history (
    HISTORY_STREAM* hs,     // history stream to
                            // prune
    DELTA_STATE* ds         // delta state at
        = NULL              // boundary of pruning
                            // returned
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: Snips the graph of DELTA_STATEs just before the given state and deletes the piece of the graph that does not include the active state. Thus one can prune forward branches by passing a state after the current state. One can prune past history by passing a state prior to the current state. It is impossible to prune away the active state.

Errors: The pointer to ds or hs is NULL.
The given delta state is not in the given history stream.

Limitations: The number of places that the two laws equal must be finite.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_query_state_validity

Function: History and Roll

Action: Returns TRUE if the given DELTA_STATE is in the HISTORY_STREAM.

Prototype:
```
outcome api_query_state_validity (
    DELTA_STATE* ds,        // input to test
    logical& change_state_possible,// TRUE if state
                            // is valid
    HISTORY_STREAM* hs      // input history stream
        = NULL
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: Returns TRUE when the given DELTA_STATE in the HISTORY_STREAM. If no HISTORY_STREAM is supplied, the default stream is used.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_radial_pattern

Function: Patterns

Action: Creates a radial pattern.

Prototype:

```
outcome api_radial_pattern (
    pattern*& pat,           // created pattern
    const SPAposition& center,// center (root)
                             // position
    const SPAvector& normal, // normal to pattern
                             // plane
    int num_radial,          // # of radial pattern
                             // rings
    int num_angular,         // # of polar pattern
                             // radii
    double spacing,          // distance between
                             // pattern rings
    const SPAvector& start   // start direction
        =*(SPAvector*)NULL_REF,
    logical not_rotate       // rotation flag
        = FALSE,
    double start_angle       // start angle
        = 0.0,
    double end_angle         // end angle
        = 2.0* 3.14159265358979323846,
    double ratio             // ratio of minor/major
        = 1.0,               // radii
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/vector.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernutil/law/pattern.hxx”
#include ”kernel/kernutil/law/pattern_api.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Creates a two–dimensional radial pattern defined by a root position center (which may or may not lie upon the seed entity) and the vector normal, which sets the orientation of the pattern. The number of elements in the radial and angular directions are specified by num_radial and num_angular, respectively, and the distance between successive rings of the pattern by the spacing argument. The optional start argument specifies the direction of the first spoke of the pattern. The elements of the pattern are kept in a fixed relative orientation if not_rotate is TRUE. The start_angle and end_angle arguments fix the angular extent of the pattern, while the ratio argument sets the ratio of minor/major radii of the pattern perimeter. The pattern coordinates are specified in the order (radial, angular).

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAposition center(0, 0, 0);
SPAvector normal(0, 0, 1);
int num_radial = 4;
int num_angular = 5;
double spacing = 3.0;
check_outcome(result = api_radial_pattern(pat,
center, normal, num_radial, num_angular, spacing));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
pat));

// Clean up
pat–>remove();
```

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_random_keep_pattern

Function: Patterns

Action: Creates a new pattern by applying a random keep–filter to an existing pattern.

Prototype:
```
outcome api_random_keep_pattern (
    pattern*& pat,           // created pattern
    const pattern& in_pattern,  // input pattern
    double keep_fraction,    // approximate fraction
                             // of elements to keep
    logical merge            // merge flag
        = TRUE,
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernutil/law/pattern.hxx”
#include ”kernel/kernutil/law/pattern_api.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Applies a periodic keep–filter to an existing pattern, merging with any existing filter or, optionally (with merge=FALSE), replacing it. The argument fraction determines the fraction of pattern elements that are kept.

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAvector x_vec(4.0, 0, 0);
int num_x = 12;
SPAvector y_vec(0, 4.0, 0);
int num_y = 12;
check_outcome(result = api_linear_pattern(pat, x_vec,
num_x, y_vec, num_y));
```

```
// Modify the pattern
pattern* mod_pat = NULL;
double keep_fraction = 0.5;
check_outcome(result =
api_random_keep_pattern(mod_pat,  *pat,
keep_fraction));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
mod_pat));

// Clean up
pat->remove();
mod_pat->remove();
```

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_random_offset_pattern

Function: Patterns

Action: Creates a new pattern by adding random offsets to an existing pattern.

Prototype:

```
outcome api_random_offset_pattern (
    pattern*& pat,          // created pattern
    const pattern& in_pat,  // input pattern
    const SPAvector& amplitude,// maximum
                            // displacements
                            // in 3 dimensions
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "baseutil/vector/vector.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernutil/law/pattern.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Creates a new pattern by adding random offsets at each site of an existing pattern. The components of the amplitude argument specify the magnitudes of the maximum offsets that are imposed in each dimension.

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAvector x_vec(4.0, 0, 0);
int num_x = 8;
SPAvector y_vec(0, 2.0, 0);
int num_y = 10;
check_outcome(result = api_linear_pattern(pat, x_vec,
num_x, y_vec, num_y));

// Modify the pattern
pattern* mod_pat = NULL;
SPAvector amplitude(1.0, 0.5, 4.0);
check_outcome(result =
api_random_offset_pattern(mod_pat, *pat, amplitude));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
mod_pat));

// Clean up
pat->remove();
mod_pat->remove();
```

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_random_orient_pattern

Function: Patterns

Action: Creates a new pattern by applying random rotations at each site of an existing pattern.

Prototype:
```
outcome api_random_orient_pattern (
    pattern*& pat,              // created pattern
    const pattern& in_pat,      // input pattern
    const SPAposition& root     // root position
        = SPAposition(0, 0, 0),
    const SPAinterval& axial_range// range of axial
        = SPAinterval(0.0,      // rotation angles
        2.0* 3.14159265358979323846),
    const SPAvector& axial_dir  // axis for tilt
        =*(SPAvector*)NULL_REF,
    const SPAinterval& tilt_range   // tilt range
        = SPAinterval(0, 3.14159265358979323846),
    const SPAvector& tilt_dir   // tilt direction
        =*(SPAvector*)NULL_REF,
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/vector/interval.hxx"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/vector.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernutil/law/pattern.hxx"
#include "kernel/kernutil/law/pattern_api.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Creates a new pattern by applying random rotations at each site of an existing pattern, using root as the position on the seed entity about which the rotation is to occur. The default arguments yield a totally random rotation. If the user specifies the tilt_dir and/or axial_dir arguments, the former gives the direction about which the interval tilt_range is applied, while the latter gives the direction about which the interval axial_range is applied. If the tilt_dir argument is not orthogonal to axial_dir, only its orthogonal component is used.

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAvector x_vec(4.0, 0, 0);
int num_x = 8;
SPAvector y_vec(0, 2.0, 0);
int num_y = 10;
check_outcome(result = api_linear_pattern(pat, x_vec,
num_x, y_vec, num_y));

// Modify the pattern
pattern* mod_pat = NULL;
SPAposition root(0, 0, 0);
SPAinterval axial_range(0, 2 * M_PI);
SPAinterval tilt_range(0, 0);
SPAvector axial_dir(1, 0, 0);
SPAvector tilt_dir(0, 1, 0);
check_outcome(result =
api_random_orient_pattern(mod_pat, *pat, axial_range,
tilt_range, axis));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
mod_pat));
```

```
// Clean up
pat->remove();
mod_pat->remove();
```

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_random_pattern

Function: Patterns

Action: Creates a random pattern within the indicated region.

Prototype:

```
outcome api_random_pattern (
    pattern*& pat,           // pattern created
    const SPAvector& extents,// vector components
                             // give pattern extents
                             // in each direction
    int num_elements,        // # of pattern elements
    int dimension            // pattern dimensionality
        = 3,
    logical ellipsoidal      // ellipsoidal flag
        = FALSE,
    const SPAvector& x_vec   // direction for first
        = SPAvector(1, 0, 0),// extent component
    const SPAvector& y_vec   // direction for second
        = SPAvector(0, 1, 0),// extent component
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "baseutil/vector/vector.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernutil/law/pattern.hxx"
#include "kernel/kernutil/law/pattern_api.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Creates a random pattern of number elements, centered at the location of the pattern seed entity and extending distances given by the components of extents in dimension dimensions. The arguments x_vec and y_vec specify the orientation of the pattern, and are the directions associated with the first two components of extents. (The third component is associated with the cross product of these arguments.) When an ellipsoidal pattern is selected (ellipsoidal=TRUE), the number of pattern elements actually generated may differ somewhat from number.

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAvector extents = (50, 25, 10);
int number = 100;
int dimensions = 3;check_outcome(result =
api_random_pattern(pat, extents, number,
dimensions));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
pat));

// Clean up
pat->remove();
```

Errors: The number of elements specified is less than one, or the dimensionality is greater than three or less than one.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_random_scale_pattern

Function: Patterns

Action: Creates a new pattern by applying a random scale to an existing pattern.

Prototype:

```
outcome api_random_scale_pattern (
    pattern*& pat,          // created pattern
    const pattern& in_pattern,// input pattern
    double min_scale,       // lower bound to the
                            // applied scale values
    double max_scale,       // upper bound to the
                            // applied scale values
    const SPAposition& root,// root position
    logical merge = TRUE,   // merge flag
    AcisOptions* ao = NULL  // acis options
    );

outcome api_random_scale_pattern (
    pattern*& pat,          // created pattern
    const pattern& in_pattern,// input pattern
    const SPAvector& min_scale,// lower bound to the
                            // applied scale values
    const SPAvector& max_scale,// upper bound to the
                            // applied scale values
    const SPAposition& root,// root position
    logical merge = TRUE,   // merge flag
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernutil/law/pattern.hxx”
#include ”kernel/kernutil/law/pattern_api.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
#include ”baseutil/vector/vector.hxx”
```

Description: Applies a random scale to an existing pattern, merging with any existing scaling or, optionally (with merge=FALSE), replacing it. The arguments min_scale and max_scale place limits upon the scale values, and can be given as vectors when nonuniform scaling is desired. which_dim specifies the dimension in which the scale is applied. The position root specifies the neutral point about which the scaling takes place (i.e., the point on the seed entity that remains fixed while the entity’s dimensions are altered). Both min_scale and max_scale must be greater than zero.

The following code snippet shows an example of how this API can be used.

```
// Create a pattern
pattern* pat = NULL;
SPAvector x_vec(4.0, 0, 0);
int num_x = 8;
SPAvector y_vec(0, 2.0, 0);
int num_y = 10;
check_outcome(result = api_linear_pattern(pat, x_vec,
num_x, y_vec, num_y));

// Modify the pattern
pattern* mod_pat = NULL;
double min_scale = 0.5;
double max_scale = 2.0;
SPAposition root(0, 0, 0);
check_outcome(result =
api_random_scale_pattern(mod_pat, *pat, min_scale,
max_scale, root));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
mod_pat));

// Clean up
pat->remove();
mod_pat->remove();
```

Errors: One or more of the specified scaling factors is zero or negative.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_remove_pattern

Function: Patterns

Action: If the input entity is patterned, removes the pattern from it and from all other associated patterned entities.

Prototype:
```
outcome api_remove_pattern (
    ENTITY* ent,            // entity to remove
                            // pattern from
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kernutil/law/pattern_api.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: System routine

api_remove_state

Function: History and Roll

Action: Merges a DELTA_STATE instance into a HISTORY_STREAM.

Prototype:
```
outcome api_remove_state (
    DELTA_STATE* ds         // state to remove
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: This API removes a DELTA_STATE from its owning HISTORY_STREAM without deleting it. This is used to in conjunction with api_note_state and api_remove_state to build multiple independent history streams. After noting a state, it can be moved to an alternate stream by removing it from the default stream, and adding it using api_add_state. To the stream it is to become a part of.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_remove_transf

Function: Transforms, Modifying Models

Action: Removes (discards) the transformation of a body.

Prototype:
```
outcome api_remove_transf (
    ENTITY* entity,          // entity of interest
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Each body contains a transformation matrix that gives the relationship between its internal coordinate system and that of the world. This API discards this transformation and places the body in the world coordinate system.

Errors: Pointer to body is NULL.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_replace_edge_with_tedge

Function: Precision and Tolerance, Tolerant Modeling

Action: Replaces an edge with a tolerant edge.

Prototype:
```
outcome api_replace_edge_with_tedge (
    EDGE* this_edge,         // edge to replace
    TEDGE*& this_tedge,      // new tolerant edge
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/top/edge.hxx″
#include ″kernel/kerndata/top/tedge.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: Replaces an edge (EDGE), its coedges (COEDGE), and its vertices (VERTEX), respectively with a tolerant edge (TEDGE), tolerant coedges (TCOEDGE), and tolerant vertices (TVERTEX).

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_replace_tedge_with_edge

Function: Precision and Tolerance, Tolerant Modeling

Action: Replaces a tolerant edge with a normal edge.

Prototype:
```
outcome api_replace_tedge_with_edge (
    TEDGE* this_tedge,       // tolerant tedge to
                             // replace
    EDGE*& this_edge,        // new edge
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/top/edge.hxx″
#include ″kernel/kerndata/top/tedge.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: Replaces a tolerant edge (TEDGE), its tolerant coedges (TCOEDGE), and its tolerant vertices (TVERTEX), respectively with a normal edge (EDGE), coedges (COEDGE), and vertices (VERTEX).

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_replace_tvertex_with_vertex

Function: Precision and Tolerance, Tolerant Modeling

Action: Replaces a tolerant vertex with a normal vertex.

Prototype:
```
outcome api_replace_tvertex_with_vertex (
    TVERTEX* this_tvertex,   // tolerant vertex to
                             // replace
    VERTEX*& this_vertex,    // new vertex
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/top/tvertex.hxx″
#include ″kernel/kerndata/top/vertex.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: Replaces a tolerant vertex (TVERTEX) with a normal vertex (VERTEX).

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_replace_vertex_with_tvertex

Function: Precision and Tolerance, Tolerant Modeling

Action: Replaces a vertex with a tolerant vertex.

Prototype:
```
outcome api_replace_vertex_with_tvertex (
    VERTEX* this_vertex,     // vertex to replace
    TVERTEX*& this_tvertex,  // new tolerant vertex
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/top/tvertex.hxx”
#include ”kernel/kerndata/top/vertex.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Replaces a vertex (VERTEX) with a tolerant vertex (TVERTEX).

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_reset_boxes

Function: Precision and Tolerance, Tolerant Modeling

Action: Removes and then adds back bounding boxes from the selected body and its subparts (or just the selected entity if it’s not a body).

Prototype:
```
outcome api_reset_boxes(
    ENTITY* ent,             // entity of interest
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/data/entity.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_restore_entity_list

Function: SAT Save and Restore

Action: Restores an entity_list from disk.

Prototype:
```
outcome api_restore_entity_list (
    FILE* file_ptr,          // open file descriptor
    logical text_mode,       // TRUE if file is text,
                             // FALSE if binary
    ENTITY_LIST& entities,   // returns entities made
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/lists/lists.hxx″
#include ″baseutil/logical.h″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: The file pointer is an open file positioned at the point where this API begins the restore entity. When the restore is complete, the file will be correctly positioned at the end of the save entity. This allows an application to restore multiple entities intermixed with other application specific data in a single save file.

Establish the calling routine whether the file is text or binary and set text_mode correctly [TRUE for SAT file or FALSE for SAB (binary) file].

Restoring a binary file is about twice as fast as restoring a text file; however, use binary files only when the file is created and read on the same version of the system running on the same type of machine.

When an entity is restored from a file, any unrecognized main entity types (BODY, CURVE, etc.) are skipped and any references to those entities are set to NULL. Unrecognized descendent entities of ATTRIB, SURFACE, or CURVE generate a new record for their immediate owner class and references to them become references to the new record. If a record for a derived class of ATTRIB is not recognized, an ATTRIB record results so that the chain of attributes for the entity owning the unrecognized attribute remains connected.

It is possible to restore entities made by versions having different sets of attribute classes. Attribute types common to the two versions are restored, but attributes of types unknown to the receiving version are ignored.

A warning is given if the version of the product receiving the model differs from the version that made the save file. It is an error if the current product is older than that recorded in the file. Errors can also occur if you use two different C runtime DLLs (e.g., one release and one debug) when using ACIS. Refer to the “C Runtime Library DLL” section in the Application Development Manual for more details.

Errors: Warning: Version number of this system differs from version that made the save file being read.

Warning: Record for unrecognized entity is being skipped. Unable to read file. Malformed save file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_restore_entity_list_file

Function: SAT Save and Restore

Action: Restores an entity_list from disk.

Prototype:
```
outcome api_restore_entity_list_file (
    FileInterface* file_ptr, // open file descriptor
    ENTITY_LIST& entities,   // returns entities
                             // restored
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/lists/lists.hxx”
#include ”kernel/kernutil/fileio/fileif.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API restores a list of entities from a file. The file_ptr points to an open file positioned at the point where this API begins the restore entity. When the restore is complete, the file will be correctly positioned at the end of the entity save. This allows an application to restore multiple entities intermixed with other application specific data in a single save file.

The calling API must establish whether the file is text or binary and text_mode must be set correctly.

Restoring a binary file is roughly twice as fast as restoring a text file. However, use binary files only when the file is created and read on the same version of ACIS running on the same type of machine. If an unrecognized entity type is encountered in a binary file, the restore process goes awry.

When an entity is restored from a text file, any unrecognized main entity types (BODY, CURVE, etc.) are skipped and any references to those entities are set to NULL. Unrecognized descendent entities of ATTRIB, SURFACE, or CURVE generate a new record for their immediate owner class and references to them become references to the new record. If a record for a derived class of ATTRIB is not recognized, at the least an ATTRIB record will result so that the chain of attributes for the entity owning the unrecognized attribute remains connected.

It is possible to restore entities made by versions of ACIS having different sets of attribute classes. Attribute types common to the two versions will be restored, but attributes of types unknown to the receiving version of ACIS will be ignored.

Reading from text files gives better recovery from error than does reading from binary files.

A warning is given if the version of ACIS receiving the model differs from the version that made the save file. It is an error if the current ACIS is older than that recorded in the file.

Errors: Warning: Version number of this ACIS differs from version that made the save file being read.

Warning: Record for unrecognized entity is being skipped. Unable to read file. Malformed save file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_restore_entity_list_with_history

Function: SAT Save and Restore, History and Roll

Action: Restores an entity_list from disk.

Prototype:
```
outcome api_restore_entity_list_with_history (
    FILE* file_ptr,             // open file
                                // descriptor
    logical text_mode,          // TRUE if file is
                                // text, FALSE if
                                // binary
    ENTITY_LIST& entities,      // returns entities
                                // made
    HISTORY_STREAM_LIST& hslist,// returns history
                                // streams made
    DELTA_STATE_LIST& dslist,       // returns delta
                                // states made
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: The file pointer is an open file positioned at the point where this API begins the restore entity. When the restore is complete, the file will be correctly positioned at the end of the save entity. This allows an application to restore multiple entities intermixed with other application specific data in a single save file.

Establish in the calling routine whether the file is text or binary and set text_mode correctly.

Restoring a binary file is about twice as fast as restoring a text file; however, use binary files only when the file is created and read on the same version of the system running on the same type of machine.

When an entity is restored from a file, any unrecognized main entity types (BODY, CURVE, etc.) are skipped and any references to those entities are set to NULL. Unrecognized descendent entities of ATTRIB, SURFACE, or CURVE generate a new record for their immediate owner class and references to them become references to the new record. If a record for a derived class of ATTRIB is not recognized, an ATTRIB record results so that the chain of attributes for the entity owning the unrecognized attribute remains connected.

It is possible to restore entities made by versions having different sets of attribute classes. Attribute types common to the two versions are restored, but attributes of types unknown to the receiving version are ignored.

A warning is given if the version of the product receiving the model differs from the version that made the save file. It is an error if the current product is older than that recorded in the file.

Application data referring to DELTA_STATEs or HISTORY_STREAMs can be restored as in the following pseudo code.

```
class app_data {
    DELTA_STATE* ds;
    void save(DELTA_STATE_LIST& dslist) {
        write_int(dslist.lookup(ds));
    }
    void restore(DELTA_STATE_LIST& dslist) {
        ds = read_int();
    }
    void fix_pointers(DELTA_STATE_LIST& dslist) {
        if( (int) < 0 ) {
            ds = NULL;
        } else {
            ds = dslist[i];
        }
    }
};
DELTA_STATE_LIST dslist;
HISTORY_STREAM_LIST hslist;
ENTITY_LIST elist;
api_restore_entity_list_with_history
    (file, TRUE, elist, hslist,dslist);
```

```
foreach(app_data* ap) {
    ap->restore(dslist);
}
foreach(app_data* ap) {
    ap->fix_pointers(dslist);
}
```

A similar procedure can be used when restoring application data that refers to history streams. See api_save_entity_list_with_history for an example of how to save the above app_data

Errors: Warning: Version number of this system differs from version that made the save file being read.

Warning: Record for unrecognized entity is being skipped. Unable to read file. Malformed save file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_restore_entity_list_with_history_file

Function: SAT Save and Restore, History and Roll

Action: Restores an entity_list from disk.

Prototype:

```
outcome api_restore_entity_list_with_history_file (
    FileInterface* file_ptr,     // open file
                                 // descriptor
    ENTITY_LIST& entities,       // returns entities
                                 // made
    HISTORY_STREAM_LIST& hslist,// returns history
                                 // streams made
    DELTA_STATE_LIST& dslist,        // returns delta
                                 // states made
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernutil/fileio/fileif.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description:

The file pointer is an open file positioned at the point where this API begins the restore entity. When the restore is complete, the file will be correctly positioned at the end of the save entity. This allows an application to restore multiple entities intermixed with other application specific data in a single save file.

Establish in the calling routine whether the file is text or binary and set text_mode correctly.

Restoring a binary file is about twice as fast as restoring a text file; however, use binary files only when the file is created and read on the same version of the system running on the same type of machine.

When an entity is restored from a file, any unrecognized main entity types (BODY, CURVE, etc.) are skipped and any references to those entities are set to NULL. Unrecognized descendent entities of ATTRIB, SURFACE, or CURVE generate a new record for their immediate owner class and references to them become references to the new record. If a record for a derived class of ATTRIB is not recognized, an ATTRIB record results so that the chain of attributes for the entity owning the unrecognized attribute remains connected.

It is possible to restore entities made by versions having different sets of attribute classes. Attribute types common to the two versions are restored, but attributes of types unknown to the receiving version are ignored.

A warning is given if the version of the product receiving the model differs from the version that made the save file. It is an error if the current product is older than that recorded in the file.

Application data referring to DELTA_STATEs or HISTORY_STREAMs can be restored as in the following pseudo code.

```
class app_data {
    DELTA_STATE* ds;
    void save(DELTA_STATE_LIST& dslist) {
        write_int(dslist.lookup(ds));
    }
    void restore(DELTA_STATE_LIST& dslist) {
        ds = read_int();
    }
    void fix_pointers(DELTA_STATE_LIST& dslist) {
        if( (int) < 0 ) {
            ds = NULL;
        } else {
            ds = dslist[i];
        }
    }
};
DELTA_STATE_LIST dslist;
HISTORY_STREAM_LIST hslist;
ENTITY_LIST elist;
api_restore_entity_list_with_history
    (file, TRUE, elist, hslist,dslist);

foreach(app_data* ap) {
    ap->restore(dslist);
}
foreach(app_data* ap) {
    ap->fix_pointers(dslist);
}
```

A similar procedure can be used when restoring application data that refers to history streams. See api_save_entity_list_with_history for an example of how to save the above app_data

Errors: Warning: Version number of this system differs from version that made the save file being read.

Warning: Record for unrecognized entity is being skipped. Unable to read file. Malformed save file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_restore_history

Function: History and Roll

Action: Restores a history stream and associated entities and entity id information from a file.

Prototype:

```
outcome api_restore_history (
    FILE* file_ptr,          // open file descriptor
    logical text_mode,       // text mode
    HISTORY_STREAM_LIST& hlist,// restored history
                             // streams
    logical create_new_hs    // flag for creating
        = FALSE,             // new history stream
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

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

Description: This routine restores entities and any associated history information from a file. If the file has no history (i.e. it was created using api_save_entity_list) then the create_new_hs flag is examined. If the flag is TRUE, a new history stream is instantiated to hold the created entities. If the flag is FALSE, the entities in the file are created in the default stream. All created history streams are added to the history stream list hlist; no change in the history stream list indicates the file had no history and the entities were placed in the default stream. api_get_active_entities can be called on the streams in hlist to find the entities read in.

The file pointer is an open file positioned at the point where this API begins the restore entity. When the restore is complete, the file will be correctly positioned at the end of the save entity. This allows an application to restore multiple entities intermixed with other application–specific data in a single save file.

Establish in the calling routine whether the file is text or binary and set text_mode correctly: TRUE if the file is text, FALSE if binary. Restoring a binary file is about twice as fast as restoring a text file. However, use binary files only when the file is created and read on the same version of the system running on the same type of machine.

When an entity is restored from a file, any unrecognized main entity types (BODY, CURVE, etc.) are skipped and any references to those entities are set to NULL. Unrecognized descendent entities of ATTRIB, SURFACE, or CURVE generate a new record for their immediate owner class, and references to them become references to the new record. If a record for a derived class of ATTRIB is not recognized, an ATTRIB record results so that the chain of attributes for the entity owning the unrecognized attribute remains connected.

It is possible to restore entities made by versions having different sets of attribute classes. Attribute types common to the two versions are restored, but attributes of types unknown to the receiving version are ignored.

A warning is given if the version of the product receiving the model differs from the version that made the save file. It is an error if the current product is older than that recorded in the file.

Errors: Warning: Version number of this system differs from version that made the save file being read.

Warning: Record for unrecognized entity is being skipped. Unable to read file. Malformed save file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_restore_history_file

Function: History and Roll

Action: Restores an history stream and associated entities and entity id information from a file.

Prototype:

```
outcome api_restore_history_file (
    FileInterface* file_ptr, // open file descriptor
    HISTORY_STREAM_LIST& hlist,// restored history
                               // streams
    logical create_new_hs    // flag for creating
        = FALSE,             // new history stream
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/kernutil/fileio/fileif.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This routine is equivalent to api_save_history, taking a FileInterface* rather than a FILE*. Please refer to the description of that routine.

If create_new_hs is set TRUE and the restored file has no history, this function will create a new history stream.

Errors: Warning: Version number of this system differs from version that made the save file being read.

Warning: Record for unrecognized entity is being skipped. Unable to read file. Malformed save file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_resume_journal

Function: ACIS Journal

Action: Sets the status flag for journalizing to on, enabling the snapshot journal mechanism.

Prototype:
```
outcome api_resume_journal (
    AcisOptions* ao          // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/acis_journal.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernapi/api/api.hxx"
```

Description: Sets the status flag to on to enable journalizing.

Errors: If this is used before start. The header will not be written and some functions will fail to journalize.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/acis_journal.hxx

Effect: System routine

api_roll_n_states

Function: History and Roll, Part Management

Action: Modifies modeler state by applying zero or more delta_states.

Prototype:
```
outcome api_roll_n_states (
    HISTORY_STREAM* hs,      // history stream to roll
    int nRequest,            // number of states to
                             // roll; positive is
                             // forward, negative is
                             // backward.
    int& nActual             // returns number of
                             // delta states rolled
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
```

Description: This API modifies the modeler's state by rolling forward or back the given number of times. When rolling forward past a branch in the history stream the branch taken is unspecified. To take a particular, save a pointer to a state on the branch and use api_change_to_state. Branches are created by rolling back and then making additional changes to the model.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_save_entity_list

Function: SAT Save and Restore, Entity, Part Management

Action: Writes a list of entities to disk as text or binary.

Prototype:

```
outcome api_save_entity_list (
    FILE* file_ptr,          // open file
                             // descriptor
    logical text_mode,       // TRUE if file is text,
                             // FALSE if binary
    ENTITY_LIST const&       // returns entities
        entity_list,         // to save
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

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

Description: The file pointer argument should be an open file positioned at the point where this API begins the entity save. When the save is complete, the file will be correctly positioned at the end of the entity save; therefore, an application can save multiple bodies intermixed with other application specific data in a single save file.

The entities are written to disk as a sequence of records, one per model entity. Writing records in binary is roughly twice as fast as writing in text and the files are some 20 per cent shorter. It is recommended you use binary save files only for short-term storage. Write and read binary files only by the same version running on the same type of hardware.

Beginning with ACIS release 6.3, it is **required** that the product ID and units be populated for the file header (using class FileInfo) before you can save a SAT file. Refer to the reference templates for the class FileInfo and function api_set_file_info for more information.

Errors can also occur if you use two different C runtime DLLs (e.g., one release and one debug) when using ACIS. Refer to the “C Runtime Library DLL” section in the Application Development Manual for more details.

Each entity record begins with a string identifier denoting its type. When a file is restored, records of unrecognized derived classes will be ignored.

The floating point precision for real numbers in text files is six digits for single precision and 15 digits for double precision.

Errors: Failed to save entities; e.g., unable to write disk file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_save_entity_list_file

Function: SAT Save and Restore

Action: Writes a list of entities to disk in text or binary format.

Prototype:
```
outcome api_save_entity_list_file (
    FileInterface* file_ptr,        // open file
                                    // descriptor
    ENTITY_LIST const& entity_list, // returns
                                    // entities to be
                                    // saved
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kerndata/lists/lists.hxx″
#include ″kernel/kernutil/fileio/fileif.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API creates the file pointer argument an open file positioned at the point where this API begins the entity save. When the save is complete, the file will be correctly positioned at the end of the entity save; therefore, an application can save multiple bodies intermixed with other application specific data in a single save file.

The entity is written to disc as a sequence of records, one per model entity. Writing records in binary is roughly twice as fast as writing in text and the files are some 20 per cent shorter. However, use binary save files only for short–term storage. Write and read binary files only by the same version of ACIS, running on the same type of hardware and with the same set of application–derived classes such as attributes.

Each entity record begins with a string identifier denoting its type. When a text file (only) is restored, records of unrecognized derived classes will be ignored. The floating point precision for real numbers in text files is six digits for single precision and 15 digits for double precision.

Beginning with ACIS release 6.3, it is **required** that the product ID and units be populated for the file header (using class FileInfo) before you can save a SAT file. Refer to the reference templates for the class FileInfo and function api_set_file_info for more information.

Errors: Failed to save entity; e.g., unable to write to disc file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_save_entity_list_with_history

Function: SAT Save and Restore, History and Roll

Action: Writes a list of entities to disk as text or binary.

Prototype:
```
outcome api_save_entity_list_with_history (
    FILE* file_ptr,              // open file
                                 // descriptor
    logical text_mode,           // TRUE if file is
                                 // text, FALSE if
                                 // binary
    ENTITY_LIST const&           // entities to
        entity_list,             // save
    HISTORY_STREAM_LIST& hslist,// history streams to
                                 // save
    DELTA_STATE_LIST& dslist,        // returns delta
                                 // states saved
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: The file pointer argument should be an open file positioned at the point where this API begins the entity save. When the save is complete, the file will be correctly positioned at the end of the entity save; therefore, an application can save multiple bodies intermixed with other application specific data in a single save file.

The entities are written to disk as a sequence of records, one per model entity. Writing records in binary is roughly twice as fast as writing in text and the files are some 20 per cent shorter however, use binary save files only for short–term storage. Write and read binary files only by the same version running on the same type of hardware.

Each entity record begins with a string identifier denoting its type. When a file is restored, records of unrecognized derived classes will be ignored.

The floating point precision for real numbers in text files is six digits for single precision and 15 digits for double precision.

History data is saved after active entities in a form allowing api_restore_entity_list to restore without history if desired.

The returned dslist can be used by the application to map DELTA_STATE pointers to unique integers and back again during save and restore to maintain an association between DELTA_STATEs and application data. For example, one might use the following pseudo code

```
class app_data {
    DELTA_STATE* ds;
    void save(DELTA_STATE_LIST& dslist) {
        write_int(dslist.lookup(ds));
    }
    void restore(DELTA_STATE_LIST& dslist) {
        ds = read_int();
    }
    void fix_pointers(DELTA_STATE_LIST& dslist) {
        if( (int) < 0 ) {
            ds = NULL;
        } else {
            ds = dslist[i];
        }
    }
};

DELTA_STATE_LIST dslist;
HISTORY_STREAM_LIST hslist;
ENTITY_LIST elist;
elist.add(entity_to_save);
api_save_entity_list_with_history
    (file, TRUE, elist, hslist,dslist);

foreach(app_data* ap) {
    ap–>save(dslist);
}
```

See api_restore_entity_list_with_history for an example of how to restore the above app_data

Some entities may have HISTORY_STREAMs attached via an ATTRIB_HISTORY. In this case the hslist would be larger on return than on entry. The returned list can be used as with the dslist when saving application data.

Beginning with ACIS release 6.3, it is **required** that the product ID and units be populated for the file header (using class FileInfo) before you can save a SAT file. Refer to the reference templates for the class FileInfo and function api_set_file_info for more information.

Errors: Failed to save entities; e.g., unable to write disk file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_save_entity_list_with_history_file

Function: SAT Save and Restore, History and Roll

Action: Writes a list of entities to disk as text or binary.

Prototype:

```
outcome api_save_entity_list_with_history_file (
    FileInterface* file_ptr,        // open file
                                    // descriptor
    ENTITY_LIST const& entity_list, // entities to
                                    // save
    HISTORY_STREAM_LIST& hslist,    // history
                                    // streams to
                                    // save
    DELTA_STATE_LIST& dslist,       // returns delta
                                    // states saved
    logical mainline_only       // save only active
        = FALSE,                // delta States
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kernutil/fileio/fileif.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description:

The file pointer argument should describe an open file positioned at the point where this API begins the entity save. When the save is complete, the file will be correctly positioned at the end of the entity save; therefore, an application can save multiple bodies intermixed with other application specific data in a single save file.

The entities are written to disk as a sequence of records, one per model entity. Writing records in binary is roughly twice as fast as writing in text and the files are some 20 per cent shorter however, use binary save files only for short–term storage. Write and read binary files only by the same version running on the same type of hardware.

Each entity record begins with a string identifier denoting its type. When a file is restored, records of unrecognized derived classes will be ignored.

The floating point precision for real numbers in text files is six digits for single precision and 15 digits for double precision.

History data is saved after active entities in a form allowing api_restore_entity_list to restore without history if desired.

The returned dslist can be used by the application to map DELTA_STATE pointers to unique integers and back again during save and restore to maintain an association between DELTA_STATEs and application data. For example, one might use the following pseudo code:

```
class app_data {
    DELTA_STATE* ds;
    void save(DELTA_STATE_LIST& dslist) {
        write_int(dslist.lookup(ds));
    }
    void restore(DELTA_STATE_LIST& dslist) {
        ds = read_int();
    }
    void fix_pointers(DELTA_STATE_LIST& dslist) {
        if( (int) < 0 ) {
            ds = NULL;
        } else {
            ds = dslist[i];
        }
    }
};

DELTA_STATE_LIST dslist;
HISTORY_STREAM_LIST hslist;
ENTITY_LIST elist;
elist.add(entity_to_save);
api_save_entity_list_with_history_file
    (fileInt, elist, hslist,dslist);

foreach(app_data* ap) {
    ap->save(dslist);
}
```

See api_restore_entity_list_with_history_file for an example of how to restore the above app_data.

Some entities may have HISTORY_STREAMs attached via an ATTRIB_HISTORY. In this case the hslist would be larger on return than on entry. The returned list can be used as with the dslist when saving application data.

Beginning with ACIS release 6.3, it is **required** that the product ID and units be populated for the file header (using class FileInfo) before you can save a SAT file. Refer to the reference templates for the class FileInfo and function api_set_file_info for more information.

Errors: Failed to save entities; e.g., unable to write disk file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_save_history

Function: SAT Save and Restore, History and Roll

Action: Writes a history stream and associated entities and entity ID information to a file.

Prototype:
```
outcome api_save_history (
    FILE* file_ptr,          // open file descriptor
    logical text_mode,       // text mode flag
    HISTORY_STREAM* hs       // history stream
        = NULL,              // to save
    logical active_ents_only// TRUE to ignore roll
        = FALSE,             // information
    logical mainline_only    // TRUE to ignore rolled
        = FALSE,             // delta states
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API saves a complete history stream to a file. History data is saved after active entities in the same format as api_save_entity_list_with_history, in a form that can be restored without history by api_restore_entity_list, if desired. If the history stream passed in is NULL, the default stream will be saved.

The active_ents_only and mainline_only flags can be used to reduce the size of the saved file. If mainline_only is TRUE, then only delta states needed to get from the beginning of the history stream to the current state (the main line) are saved. This is equivalent to ignoring all rolled delta states. If active_ents_only is TRUE, then the active entities from hs are saved, along with history stream information containing only create bulletins for these entities. active_ents_only is a more stringent condition than mainline_only; the mainline_only flag has no effect when active_ents_only is TRUE.

Unhooked annotations are not considered active entities by this API; rather they are stored with the history data. This means that they will not be restored when using api_restore_entity_list and will not be saved at all (even in the history data) when active_ents_only is TRUE. If a user wants unhooked annotations to be saved in the active entities section, he should find the unowned active entities using api_get_active_entities (with unowned_only = TRUE) and then call api_save_entity_list_with_history.

In addition to managing roll information, history streams also manage entity id information; an entity id is unique to a history stream. Entity IDs can only only be persisted by saving with history (using api_save_history or api_save_entity_list_with_history). Entity IDs are ignored when reading in with api_restore_entity_list; they are only restored when using api_restore_entity_list_with_history or api_restore_history. The main difference between using api_save_entity_list and api_save_history with active_ents_only = TRUE is that api_save_history maintains the tag information.

The file pointer argument should be an open file positioned at the point where this API begins the entity save. When the save is complete, the file will be correctly positioned at the end of the entity save. An application can save multiple histories intermixed with other application specific data in a single save file.

The entities are written to a file as a sequence of records, one per model entity. Writing records in binary is roughly twice as fast as writing in text and the files are some 20 per cent shorter. It is recommended you use binary save files only for short–term storage. Write and read binary files only by the same version running on the same type of hardware.

Errors can also occur if you use two different C runtime DLLs (e.g., one release and one debug) when using ACIS. Refer to the “C Runtime Library DLL” section in the *3D ACIS Application Development Manual* for more details.

Each entity record begins with a string identifier denoting its type. When a file is restored, records of unrecognized derived classes will be ignored.

The floating point precision for real numbers in text files is six digits for single precision and 15 digits for double precision.

Errors: Failed to save entities; e.g., unable to write disk file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_save_history_file

Function: SAT Save and Restore, History and Roll

Action: Writes a history stream and associated entities and entity id information to a file.

Prototype:
```
outcome api_save_history_file (
    FileInterface* file_ptr,// open file descriptor
    HISTORY_STREAM* hs      // stream to save
        = NULL,
    logical active_ents_only// TRUE to ignore roll
        = FALSE,            // information
    logical mainline_only   // TRUE to ignore rolled
        = FALSE,            // delta states
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/kernutil/fileio/fileif.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This routine is equivalent to api_save_history, taking a FileInterface* rather than a FILE*. Please refer to the description of that routine.

Errors: Failed to save entities; e.g., unable to write disk file.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_save_state

Function: SAT Save and Restore

Action: Save the current state of global variables into a text file.

Prototype:
```
outcome api_save_state (
    FILE* file_ptr,          // file descriptor
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API saves the current states of global variables such as options and tolerances into a text file. You may use this function to save the state to a file in your own application and load the state through the file to Scheme AIDE to compare the behaviors between your application and the test applications.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_save_version

Function: SAT Save and Restore

Action: Sets the save file format.

Prototype:
```
outcome api_save_version (
    int major_version,       // major version
    int minor_version        // minor version
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kernapi/api/kernapi.hxx″
```

Description: This API sets the output file format. For Release 1.5 and above, the system can output data in a format that a previous version can read. This is only true for objects that are compatible in the previous release.

Note *For the major version starting 4 and above, the minor version does not have any effect and setting them to zero would allow the SAT files to be read across all the minor versions for the series.*

Errors: None

Limitations: New functionality or structures in the higher release are not correctly handled by the modeler, and therefore, are not supported.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read-only

api_set_acis_options

Function: ACIS Journal

Action: Copies the AcisJournal and AcisVersion Objects from the arguments to the data members inside AcisOptions.

Prototype:
```
outcome api_set_acis_options (
    AcisOptions* ao,                // acis options
    AcisJournal& aj                 // acis journal
        = *(AcisJournal*)NULL_REF,  // to be copied
    AcisVersion& av                 // acis version
        = *(AcisVersion*)NULL_REF   // to be copied
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/version/vers.hxx”
#include ”kernel/kernapi/api/acis_journal.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
#include ”kernel/kernapi/api/api.hxx”
```

Description: Takes the arguments and copies them into the data members contained in the AcisOptionsInternal data member.

Errors: None

Limitations: The version and journal objects are true copied, they are independent from the ones contained in the AcisOptions object.

Library: kernel

Filename: kern/kernel/kernapi/api/acis_options.hxx

Effect: System routine

api_set_dbl_option

Function: Modeler Control

Action: Sets the value of the specified option to the given double.

Prototype:
```
outcome api_set_dbl_option (
    char const* name,        // name of option
    double value             // double value to set
    );
```

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

Description: This API sets the named option to the specified value. Refer to the option:list Scheme extension for a list of the available options.

Errors: NULL or unknown option name specified.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_set_default_history

Function: History and Roll

Action: Sets the input HISTORY_STREAM to be the default the history stream.

Prototype:
```
outcome api_set_default_history (
    HISTORY_STREAM* hs       // input history stream
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kerndata/bulletin/bulletin.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_set_file_info

Function: SAT Save and Restore

Action: Sets required header info to be written to ACIS save files.

Prototype:
```
outcome api_set_file_info (
    unsigned long,          // mask indicating fields
                            // to set
    FileInfo const& info    // info to be set
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/savres/fileinfo.hxx"
```

Description: The API sets the information to be written to the header of later saved files. Does not alter the model. Beginning with ACIS release 6.3, it is **required** that the product ID and units be populated for the file header before you can save a SAT file.

The mask value indicates which values in the supplied FileInfo structure are to be set. It is composed by ORing together mask values as indicated below.

The FileInfo structure contains the following fields which can be set:

product_id . . . . . Mask = FileId
String indicating the product and version that produced the save file.

units . . . . . Mask = FileUnits
Modeling units specified as millimeters per unit.

units values for common modeling units are:

–1.0 . . . . . = Units not specified
1.0 . . . . . = Millimeters
10.0 . . . . . = Centimeters
1000.0 . . . . . = Meters
1000000.0 . . . . . = Kilometers
25.4 . . . . . = Inches
304.8 . . . . . = Feet
914.4 . . . . . = Yards
1609344.0 . . . . . = Miles

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_set_int_option

Function: Modeler Control

Action: Sets the value of the specified option to the given integer.

Prototype:
```
outcome api_set_int_option (
    char const* name,       // name of option
    int value               // integer value to set
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
```

Description: This API sets the named option to the specified value. Refer to the option:list Scheme extension for a list of the available options.

Errors: NULL or unknown option name specified.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_set_journal

Function: ACIS Journal

Action: Copies the AcisJournal object to AcisOptions.

Prototype:
```
outcome api_set_journal (
    AcisOptions* ao,        // acis options
    AcisJournal& aj         // acis journal to be
                            // copied
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/acis_journal.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernapi/api/api.hxx"
```

Description: Takes the AcisJournal object and copies it into the data member contained in the AcisOptionsInternal data member.

Errors: None

Limitations: The journals object is true copied, it is independent from the one contained in the AcisOptions object.

Library: kernel

Filename: kern/kernel/kernapi/api/acis_journal.hxx

Effect: System routine

api_set_journal_name

Function: ACIS Journal

Action: Sets the snapshot journal file name.

Prototype:
```
outcome api_set_journal_name (
    AcisJournal* aj,        // acis journal
    char* name              // journal file name
    );

outcome api_set_journal_name (
    AcisOptions* ao,        // acis options
    char* name              // journal file name
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/acis_journal.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernapi/api/api.hxx"
```

Description: Sets the file name for the snapshot journal. The extension name is not needed.

Errors: None

Limitations: If a name other than AcisJour is set, then the new name would be used always. In this case, instead of serializing the output name (AcisJour_x,scm, x = 0...n), it would create always the same file (e.g. My_name.scm). This can be really useful because it will not create a large number of files if the journal is implemented in a function that is called many times and it is desired to keep only the last call (e.g. when an error occurs).

Library: kernel

Filename: kern/kernel/kernapi/api/acis_journal.hxx

Effect: System routine

api_set_str_option

Function: Modeler Control

Action: Sets the value of the specified option to the given string.

Prototype:
```
outcome api_set_str_option (
    char const* name,        // name of option
    char const* value        // double value to set
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
```

Description: This API sets the named option to the specified value. Refer to the option:list Scheme extension for a list of the available options.

Errors: NULL or unknown option name specified.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_set_version

Function: ACIS Journal

Action: Copies the version object into the acis option.

Prototype:
```
outcome api_set_version(
    AcisOptions* ao,        // acis options
    AcisVersion& av         // acis version to set
    );
```

Includes:
```
#include ”kernel/kernapi/api/acis_options.hxx”
#include ”baseutil/version/vers.hxx”
#include ”kernel/acis.hxx”
#include ”kernel/kernapi/api/api.hxx”
```

Description: Makes a true copy of the version object into the AcisOptions object.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/acis_options.hxx

Effect: Read-only

api_spherical_pattern

Function: Patterns

Action: Creates a spherical pattern.

Prototype:
```
outcome api_spherical_pattern (
    pattern*& pat,          // created pattern
    const SPAposition& center,// pattern center
    int num_latitudes,      // # of latitudinal rings
                            // in the pattern
    const SPAposition& root,// position mapped to
                            // pattern sites
    double spacing          // desired spacing for
        = 0.0,              // pattern elements
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/vector/position.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernutil/law/pattern.hxx”
#include ”kernel/kernutil/law/pattern_api.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Creates a two–dimensional spherical pattern about the center position, with the pattern seed entity at one pole of the associated sphere. The pattern elements are approximately equally spaced, with the parameter num_latitudes specifying by default the number of latitudinal rings in the pattern. If num_latitudes is set to zero, this number is instead determined by the optional spacing parameter. (This number must be specified if num_latitudes is zero.) The root position of the pattern is given by root. The pattern coordinates for spacing are specified in the order (longitude, latitude).

Errors: Both the spacing and the number of latitudes is zero.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_stackmon_limit

Function: Modeler Control

Action: Sets the limit in bytes of how much stack ACIS may use.

Prototype:
```
outcome api_stackmon_limit (
    size_t limit            // bytes of stack memory
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
```

Description: ACIS can monitor the size of the stack. This function sets the limit in bytes of how much stack ACIS may use. If the limit is exceeded, ACIS will trap, returning EXCESSIVE_RECURSION . Passing 0 results in no stack monitoring.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_start_journal

Function: ACIS Journal

Action: Sets the status flag for journalizing to on and initializes journal.

Prototype:
```
outcome api_start_journal (
    AcisOptions* ao          // acis options such as
                             // journal, version
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/acis_journal.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
#include "kernel/kernapi/api/api.hxx"
```

Description: Sets the status flag on to enable journalizing. It creates the journal file and writes down the header. It also generates the SAT file name and sets the file name counters properly.

Errors: Starting again an already enabled journal may cause erasing previously saved files or leaving incomplete journaled files.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/acis_journal.hxx

Effect: System routine

api_start_modeller

Function: Modeler Control

Action: Starts the modeler.

Prototype:
```
outcome api_start_modeller (
    int                      // memory size
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
```

Description: This API starts the modeler, defines some global variables, and does a simple check on whether static initializers have been called (a problem for non-C++ application developers). This API must be called before a call to any other API.

The argument to this API specifies how much memory to allocate for the application. If this argument is zero, the application uses as much memory as is needed.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_stop_modeller

Function: Modeler Control

Action: Terminates modeler and releases memory.

Prototype:
```
outcome api_stop_modeller ();
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
```

Description: This API attempts to release all memory allocated by ACIS. The application should not attempt to reference any data returned by earlier calls to APIs or Direct Interface routines after calling api_stop_modeller.

No other APIs should be called until api_start_modeller is called again.

This function returns a non–zero FREELIST_IN_USE outcome indicating remaining memory allocations in the internal freestore. This is due to the size-based freelist strategy implemented in the MMGR component, which allows global object constructors to use the internal heap. The memory is not returned to freestore until the objects are destructed, which occurs after program execution and consequently after api_stop_modeller. Directly after an api_start_modeller call and prior to an api_stop_modeller call, compare the number of objects already in freestore with the count still remaining. The current count of committed objects in ACIS internal freestore is returned by the check_free_lists function defined in mmgr/mmgrhusk/freelist.hxx.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_str_to_law

Function: Laws

Action: Creates a law from a string and an optional array of law data.

Prototype:
```
outcome api_str_to_law (
    const char* str,         // string of law to be
                             // created
    law** answer,            // array of supporting
                             // data used in the law
                             // creation
    law_data** data          // size of the law
        = NULL,              // data array
    int size                 // returns created law
        = 0,
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "lawutil/law_base.hxx"
#include "lawutil/law_data.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API parses a character string (str), generates the associated law classes, and returns a pointer to the top-level law that was created (answer). Deriving the law class and all associated classes individually is possible. However, it is more likely that api_str_to_law and law string parsing will be employed, because it is easier and more straightforward to implement.

The valid syntax for the character strings (str) in the law mathematical functions are given in the law symbol templates. The law mathematical functions support nesting of law symbols. Once the character string (str) has been created, it is passed to api_str_to_law along with a pointer to an output law (answer), an array of law data (data), and the size of the law data array (size).

The unary_law, binary_law, and multiple_law classes are used if the application is passing only laws into a law class, in which case it becomes a pointer to a law or an array of pointers to laws, respectively. Numbers, positions, parametric positions, vectors, and vector fields, in addition to the law symbols, are passed as input to the api_str_to_law and become laws for these purposes.

On the other hand, the unary_data_law and multiple_data_law classes are used if the application is passing more complicated structures into a law class. These could be curves, wires, surfaces, transforms, or even laws. Instead of having a pointer to a law or an array of pointers to laws, the unary_data_law and multiple_data_law classes have a pointer to a law_data class or an array of pointers to law_data classes, respectively.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_surface_pattern

Function: Patterns

Action: Creates a pattern parallel to a surface.

Prototype:

```
outcome api_surface_pattern (
    pattern*& pat,           // created pattern
    FACE* in_face,           // guide face
    int num_u,               // u-direction elements
    int num_v,               // v-direction elements
    const SPAposition& root, // position mapped
                             // to the pattern sites
    logical on_boundary      // flag to begin and end
        = FALSE,             // on face boundary
    const SPAvector& u_dir   // direction mapped to
        =*(SPAvector*)NULL_REF,// u-direction
    const SPAvector& v_dir   // direction mapped to
        =*(SPAvector*)NULL_REF,// v-direction
    AcisOptions* ao = NULL   // acis options
    );
```

```
outcome api_surface_pattern (
    pattern*& pat,           // created pattern
    const surface& in_surf,  // guide surface
    const SPApar_box& face_range,// range of surface
    int num_u,               // u–direction elements
    int num_v,               // v–direction elements
    const SPAposition& root,// position mapped
                             // to the pattern sites
    logical on_boundary      // flag to begin and end
        = FALSE,             // on face boundary
    const SPAvector& u_dir   // direction mapped to
        =*(SPAvector*)NULL_REF,// u–direction
    const SPAvector& v_dir   // direction mapped to
        =*(SPAvector*)NULL_REF,// v–direction
    const SPAtransf& in_trans// input
        =*(SPAtransf*)NULL_REF,// transform
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include ″kernel/acis.hxx″
#include ″baseutil/logical.h″
#include ″baseutil/vector/param.hxx″
#include ″baseutil/vector/position.hxx″
#include ″baseutil/vector/transf.hxx″
#include ″baseutil/vector/vector.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/face.hxx″
#include ″kernel/kerngeom/surface/surdef.hxx″
#include ″kernel/kernutil/law/pattern.hxx″
#include ″kernel/kernutil/law/pattern_api.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: Creates a two–dimensional pattern of num_u by num_v elements, equally spaced in parameter space, upon the surface and parameter range indicated either by in_surf and param_range or by in_face. The argument root specifies the position (which can be on or off the pattern seed entity, as desired) to be mapped to the pattern sites. The pattern can be extended to the face boundary by setting on_boundary to TRUE. By default, pattern members are oriented identically to one another, but will follow the surface normal if u_dir and v_dir are given. In that case, these vectors specify the directions, relative to the seed entity, that are mapped to the u– and v–directions of the face.

The following code snippet shows an example of how this API can be used.

```
// Create a hemispherical surface
FACE* face = NULL;
SPAposition origin(0, 0, 0);
double radius = 20.0;
double lo_start = 0.0;
double lo_end = 90.0;
double la_start = –360.0;
double la_end = 360.0;
SPAvector normal(0, 1, 1);
check_outcome(result = api_face_sphere(origin,
radius, lo_start, lo_end, la_start, la_end, &normal,
face));
const surface& surf = face–>geometry()–>equation();
SPApar_box param_range;
sg_get_face_par_box(face, param_range);

// Create a pattern
pattern* pat = NULL;
int u_num = 8;
int v_num = 6;
SPAposition root(0, 0, 0);
check_outcome(result = api_surface_pattern(pat, surf,
param_range, u_num, v_num, root));

// Create a prism
BODY* prism = NULL;
double height = 1.0;
double maj_rad = 1.0;
double min_rad = 0.5;
int num_sides = 3;
check_outcome(result = api_make_prism(height,
maj_rad, min_rad, num_sides, prism));

// Apply the pattern to the prism
check_outcome(result = api_set_entity_pattern(prism,
pat));

// Clean up
pat–>remove();
check_outcome(result = api_del_entity(face));
```

Errors: The number of u– or v–values is less than one, or u_dir is specified without specifying v_dir (or vice–versa), or a NULL face is given.

Limitations: None

Library: kernel

Filename: kern/kernel/kernutil/law/pattern_api.hxx

Effect: Changes model

api_terminate_kernel

Function: Modeler Control, Entity, Model Geometry, Model Topology, Construction Geometry

Action: Terminates the kernel library.

Prototype:
```
outcome api_terminate_kernel ();
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: System routine

api_terminate_spline

Function: Modeler Control, Spline Interface

Action: Terminates the spline library.

Prototype:
```
outcome api_terminate_spline ();
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/spline/api/spl_api.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/spline/api/spl_api.hxx

Effect: System routine

api_test_deep_copy

Function: Model Geometry, Model Object

Action: Test the deep copy functionality for improper sharing.

Prototype:
```
outcome api_test_deep_copy (
     ENTITY_LIST const& entity_list,// list of
                                    // entities to be
                                    // deep copied
    double numerical_tolerance  // tolerance for real
        = SPAresnor,            // value comparisons
    logical report_all_errors   // flag to skip
        = FALSE,                // attributes not
                                // deep copyable
    char* file1             // file of entities saved
        = NULL,             // before deep-copy
    char* file2             // file of entities saved
        = NULL,             // after deep-copy
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API deep copies the given list of entities. These entities are saved to SAT files and restored, and then saved out again before and after a deep copy. If this function is called in a debug build, the memory from the original will be pattern filled for additional checking of no sharing after a deep copy. A comparison is done between the two SAT files created, original.sat and deep_copy.sat.

This function is used primarily for internal testing. However, if derived entities are used outside of ACIS, this function can be used to test their deep copy capabilities.

Errors: None

Limitations: Refer to Description.

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_transform_entity

Function: Model Geometry, Transforms, Entity, Modifying Models

Action: Applies a transformation to an entity.

Prototype:
```
outcome api_transform_entity (
    ENTITY* ent,            // entity to transform
    const SPAtransf& tform, // transform to apply
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: If the entity is a BODY, this API concatenates the transform with any transform that is already applied. If the entity is not a body, this API transforms the geometry.

If the Operators Component is linked into the executable, one can do non–uniform scaling using space warping. For each library your application links in, call the API that initializes that library. Lowest libraries (like Kernel) go first.

Errors: A NULL pointer to an entity is specified.
An attempt is made to transform an entity that belongs to another entity.

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_unhook_annotations

Function: Feature Naming

Action: Traverses the active list of annotations and removes associated ATTRIB_ANNOTATIONs.

Prototype:
```
outcome api_unhook_annotations (
    is_fun is_function      // Function pointer to
        = is_ANNOTATION,    // the type of annotation
    BULLETIN_BOARD* bb      // obsolete, ignored
        = NULL,
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/bulletin/bulletin.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: The function for is_fun defaults to is_ANNOTATION. However, any is function for a class can be used. So, for example, to get the top vertex annotations from a sweep operation, this function can be passed is_SWEEP_ANNO_VERTEX_TOP as an argument.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_update_tolerance

Function: Precision and Tolerance, Tolerant Modeling

Action: Updates the tolerance on an entity.

Prototype:
```
outcome api_update_tolerance (
    ENTITY* this_entity,    // entity with tolerance
    logical& updated,       // result TRUE is a
                            // tolerant entity
                            // updated
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kerndata/data/entity.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This function calculates and updates the tolerant topology of the given entity and its children.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Read–only

api_wcs_create

Function: Work Coordinate Systems

Action: Creates a new working coordinate system.

Prototype:
```
outcome api_wcs_create (
    const SPAposition& origin,// origin of WCS
    const SPAposition& xpt,  // position on x-axis
    const SPAposition& ypt,  // position in positive
                             // xy-plane
    WCS*& new_wcs,           // returns created WCS
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "kernel/geomhusk/wcs.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates a new working coordinate system (new_wcs) with its origin at origin, its *x*-axis pointing toward xpt, and its *y*-axis pointing toward ypt.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_wcs_get_active

Function: Work Coordinate Systems

Action: Gets the active working coordinate system.

Prototype:
```
outcome api_wcs_get_active (
    WCS*& active_wcs,        // returns active WCS
                             // or NULL
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”kernel/geomhusk/wcs.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kernapi/api/kernapi.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model

api_wcs_set_active

Function: Work Coordinate Systems

Action: Sets a specified working coordinate system to be active.

Prototype:
```
outcome api_wcs_set_active (
    WCS* new_active,         // WCS to make active or
                             // NULL (model space)
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "kernel/geomhusk/wcs.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/kernapi.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: kernel

Filename: kern/kernel/kernapi/api/kernapi.hxx

Effect: Changes model