Chapter 4.
Functions

Topic: Ignore

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

api_accurate_bs3_approximation

Function: Model Geometry

Action: Recomputes the 3D B-spline surface approximation, using the newer algorithm.

Prototype:
```
outcome api_accurate_bs3_approximation (
    FACE* face,                 // input face
    double requested_tol,       // requested
                                // fit tolerance
    bs3_surface& fit_sur        // output
    =*(bs3_surface*) NULL_REF,  // bs3_surface
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "kernel/spline/bs3_srf/bs3surf.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Recomputes the 3D B-spline surface approximation (to a requested tolerance), always using the newer, more robust approach. It also removes internal double knots whenever possible. If the requested tolerance is non-positive, SPAresfit is used.

The 3D B-spline surface is returned, if requested. It is still owned by the face surface, however.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_body

Function: Construction Geometry, Model Topology, Model Object

Action: Creates an empty body.

Prototype:
```
outcome api_body (
    BODY*& bodyOut,          // empty body returned
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: Refer to Action.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_body_mass_pr

Function: Physical Properties

Action: Determines the mass properties of a body.

Prototype:

```
outcome api_body_mass_pr (
    BODY* body,                          // body to be
                                         // examined
    SPAposition const&                   // root point of
        root_proj_pl,                    // projection
                                         // plane
    SPAunit_vector const&                // normal of
        normal_proj_pl,                  // projection
                                         // plane
    int selector,                        // properties
                                         // computed
    double req_rel_accy,                 // requested
                                         // relative
                                         // accuracy
    double& volume,                      // returned
                                         // volume
    SPAposition& cofg,                   // returned
                                         // center
                                         // of gravity
    tensor& inertia,                     // returned
                                         // tensor (3 X 3)
    double p_moments[],                  // returned
                                         // moments of
                                         // inertia
    SPAunit_vector p_axes[],             // returned axes
                                         // for moments
    double& est_rel_accy_achieved,       // relative error
    double sheet_thickness = 0.0,        // thickness to
                                         // apply to
                                         // double sided
                                         // sheets
    AcisOptions* ao = NULL               // acis options
    );
```

```
outcome api_body_mass_pr (
    BODY* body,                         // body to be
                                        // examined
    int selector,                       // properties
                                        // computed
    double req_rel_accy,                // requested
                                        // relative
                                        // accuracy
    double& volume,                     // returned
                                        // volume
    SPAposition& cofg,                  // returned
                                        // center
                                        // of gravity
    tensor& inertia,                    // returned
                                        // tensor (3 X 3)
    double p_moments[],                 // returned
                                        // moments of
                                        // inertia
    SPAunit_vector p_axes[],            // returned axes
                                        // for moments
    double& est_rel_accy_achieved,      // relative error
    double sheet_thickness = 0.0,       // thickness to
                                        // apply to
                                        // double sided
                                        // sheets
    AcisOptions* ao = NULL              // acis options
    );
```

Includes:

```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/body.hxx″
#include ″kernel/kernutil/tensor/tensor.hxx″
#include ″baseutil/vector/position.hxx″
#include ″baseutil/vector/unitvec.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description:

This API finds the volume, center of gravity and moments of inertia of the given body. The center of gravity is calculated with respect to the world coordinate system. The moments of inertia are also calculated with respect to the world coordinate system, which means that the first and second moments will change if the body is transformed.

The api_body_mass_pr function has two overloaded versions. One allows the user to specify a root point and normal direction for the projection plane, the other chooses an appropriate plane automatically.

The principle axes are defined with respect to the world coordinate system. The direction of each axis is defined by a unit vector. Each axis should be orthogonal to the other two axes.

The principal moments of inertia are with respect to the coordinate system defined by the center of gravity and the principal axes. (These three values should be relatively constant under translation and rotation transformations, although their order may change depending on the principal axes.)

The major diagonal of the inertia tensor returned contains values for the inertias about each axis. For the inertia about the *x*-axis:

```
(integral((y*y + z*z)dV)
```

the off-diagonal terms are for the products of inertia; for example:

```
(integral)(x*y)dV)
```

The projection plane is specified by a point and normal (in global body space). For speed, choose the plane so that as many faces as possible project on to it as lines; i.e., are edge-on to it. To improve the accuracy of the result, set the plane to pass through a point within the body; generally, this is the mid-point of its box.

The selector determines which properties are computed:

0 All properties (volume, center of gravity, and moments of inertia) are calculated.
1 Only volume and center of gravity are calculated.
2 Only volume is calculated.

Set the requested accuracy to the desired relative error as a percentage. For example, entering 1 requests a maximum error of one percent. The application should find the mass properties for several requested accuracies and compare the results since there is a limitation to the current algorithm in that it does have a bound on how precisely the algorithm can calculate mass properties, due to hard coded convergence criteria in the functions. If one tightens the relative error value and the mass properties values don't change, that means it is as close as can be achieved.

The values for mass properties would be expected to be different depending on the normal to the projection plane for many types of objects. One should attempt to select a projection plane that will simplify the body as much as possible. For instance, if the body were an extrusion, one would choose a plane perpendicular to the extrusion direction. This makes the numerical integration simpler.

Generally, double–sided (sheet or embedded) faces are ignored when calculating mass properties – The assumption being that a sheet of zero thickness has a zero volume. This behavior can be altered by changing the value of the sheet_thickness argument from its default of 0.0 to a positive value. This will cause api_body_mass_pr to treat double–sided sheets in the body as being approximations to thin–shell bodies with the given thickness, and contributions to the mass properties will be calculated for them also. Some caveats apply when using this:

1) The approximation is only valid for 'small' values of the thickness. The range of thickness that can be considered 'small' is application dependent.
2) double–sided faces that meet non–tangentially or intersect will introduce errors as there is a double contribution to the mass properties from those regions of the faces that are within the thickness value of each other. The inside/outside flag for double–sided faces is taken into account, so a double–sided face embedded in a solid will reduce the volume if it is given a thickness value. Note that this means that an isolated 'inside' double–sided sheet will have a negative volume.

Although computation of mass properties does not make substantive changes to a model, boxes may be found and set into the model, creating a bulletin board. To make the process "read-only" in the application, call api_note_state before calling api_body_mass_pr, then call the following to roll the model back to its state before api_body_mass_pr was called:

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

Errors: The pointer to a body is NULL or does not point to a BODY.
A zero-length normal is given.
A negative accuracy is requested.
An invalid selector value is specified.
A negative sheet thickness is specified.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_body_to_2d

Function: Model Topology

Action: Converts single-sided faces to double-sided faces.

Prototype:
```
outcome api_body_to_2d (
    BODY* body,              // body to be converted
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: This API converts all the single-sided faces in a given body into double-sided faces, which converts all the shells to lumps.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_build_wire

Function: Construction Geometry, Model Topology

Action: Creates a wire from an array of positions and an array of pointers to curves.

Prototype:
```
outcome api_build_wire (
    BODY* given_body,        // body to add wire to
    logical closed,          // builds a closed wire
                             // if TRUE
    int length,              // number of spans and
                             // end positions
    SPAposition points[],    // array of positions
    curve* curves[],         // array of pointers to
                             // span curves
    BODY*& body,             // returned wire body
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/body.hxx"
#include "kernel/kerngeom/curve/curdef.hxx"
#include "baseutil/logical.h"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API constructs a wire from an array of positions and an array of pointers to curves. If the wire is closed, set closed to TRUE, and both arrays are the given length. If the wire is open, the array of curves is length –1 long, and the array of positions is the given length.

The API does not determine if the wire is self-intersecting.

If the given body is NULL, the API makes a new body. It leaves the new wire as the first entry in the wire list of the body, new or old.

The API returns the body in the last argument.

Errors: The pointer to given_body is not to a BODY.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_closed_wire

Function: Construction Geometry, Object Relationships

Action: Determines if a wire or a single-wire body is closed.

Prototype:
```
outcome api_closed_wire (
    BODY* body,              // wire body to
                             // be examined
    AcisOptions* ao = NULL   // acis options
    );

outcome api_closed_wire (
    WIRE* wire,              // wire to
                             // be examined
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: The API returns a successful outcome if the wire is closed. It does not alter the model.

This function is overloaded. This version of the API will be removed in a future release. The form of api_closed_wire that takes a WIRE instead of a BODY should be used instead.

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

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Read-only

api_create_point

Function: Construction Geometry, Model Geometry

Action: Creates a point entity at the specified position.

Prototype:
```
outcome api_create_point (
    const SPAposition& pos,  // position of point
    APOINT*& pnt,            // created point returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/geom/point.hxx″
#include ″baseutil/vector/position.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API creates an instance of APOINT with a use count of 1.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_create_text

Function: Construction Geometry, Text

Action: Creates a text entity at given position.

Prototype:
```
outcome api_create_text (
    const SPAposition& location,// baseline position
                                // of first character
    const char* string,         // string to be
                                // displayed
    const char* font,           // font name
    int size,                   // font size in
                                // points
    TEXT_ENT*& text,            // created text
                                // entity returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/geomhusk/text.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates a text entity at given location, with a given text string, font, and size.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_arc

Function: Construction Geometry, Model Geometry

Action: Creates a circular arc given the center, radius, and start and end angles.

Prototype:
```
outcome api_curve_arc (
    const SPAposition& center,  // center of arc
    double radius,           // radius of arc
    double start_angle,      // start angle in radians
    double end_angle,        // end angle in radians
    EDGE*& arc,              // created arc returned
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: This API defines an arc parallel to the *xy*-plane of the active WCS, given a center, a radius, a start_angle, and an end_angle.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_arc_3curve

Function: Construction Geometry, Model Geometry

Action: Creates a circle tangent to three curves.

Prototype:
```
outcome api_curve_arc_3curve (
    const entity_with_ray& crv1,// first curve plus
                                // approximate
                                // tangent
    const entity_with_ray& crv2,// second curve plus
                                // approximate
                                // tangent
    const entity_with_ray& crv3,// third curve plus
                                // approximate
                                // tangent
    logical full,               // TRUE (full
                                // circle) or FALSE
                                // (circular arc)
    EDGE*& arc,                 // created circle
                                // returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/geomhusk/entwray.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates an edge defined by a circular arc tangent to three curves, given three curves (crv1, crv2, and crv3), and a specification whether the curve is a full circle (logical).

Errors: The three curves must all lie in a common plane.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_arc_3pt

Function: Construction Geometry, Model Geometry

Action: Creates a circle or circular arc through three positions.

Prototype:
```
outcome api_curve_arc_3pt (
    const SPAposition& pt1,  // first position
    const SPAposition& pt2,  // second position
    const SPAposition& pt3,  // third position
    logical full,            // TRUE (full circle) or
                             // FALSE (circular arc)
    EDGE*& arc,              // created arc returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/edge.hxx″
#include ″baseutil/logical.h″
#include ″baseutil/vector/position.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API creates an arc passing through three locations. pt1 specifies a starting position and, with the *x*-axis, defines the arc's start angle. pt2 specifies a the second position. pt3 specifies the end position. full specifies whether the arc is complete or partial.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_arc_center_edge

Function: Construction Geometry, Model Geometry

Action: Creates a circle or circular arc given a center and two edge points.

Prototype:
```
outcome api_curve_arc_center_edge (
    const SPAposition& center,  // center position
    const SPAposition& pt1,  // start position
    const SPAposition& pt2,  // end position
    const  SPAunit_vector* norm,// optional normal
                             // vector
                             // or NULL
    EDGE*& arc,              // created arc returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/unitvec.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates a circle or circular arc given a center and two edge points. If the optional norm is non-NULL, the arc is created in the plane defined by center and norm. The projections of pt1 and pt2 onto this plane determine the start and end of the arc. If norm is not specified, the arc is created in the plane defined by the first of the following cases that defines a valid plane normal vector (where zaxis and yaxis are the *z*-axis and *y*-axis of the active WCS):

1. The plane defined by points.

   ```
   (pt1–center) * (pt2–center)
   ```

2. (Usually) the *xy*-plane of active WCS.

   ```
   ((pt1–center) * zaxis) * (pt1–center)
   ```

3. The *xz*-plane of active WCS.

   ```
   ((pt1–center) * yaxis) * (pt1–center)
   ```

If pt1 equals pt2, this API creates a full circle.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_arc_diagonal

Function: Construction Geometry, Model Geometry

Action: Creates a circle or circular arc given two points on the diagonal.

Prototype:

```
outcome api_curve_arc_diagonal (
    const SPAposition& pt1,  // first point on
                             // diagonal
    const SPAposition& pt2,  // second point on
                             // diagonal
    logical full,            // TRUE (full circle) or
                             // FALSE (circular arc)
    EDGE*& arc,              // created arc returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "baseutil/logical.h"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates a full or partial arc passing through two positions. The diameter of the arc is defined by the start and end positions. full determines whether the arc is complete or partial. The arc is created in the plane defined by the first of the following cases that defines a valid plane normal vector (where zaxis and yaxis are the *z*-axis and *y*-axis of the active WCS):

If pt1 equals pt2, this API creates a full circle.

1. The plane defined by points.

   ```
   (pt1–center) * (pt2–center)
   ```

2. (Usually) the *xy*-plane of active WCS.

   ```
   ((pt1–center) * zaxis) * (pt1–center)
   ```

3. The *xz*-plane of active WCS.

   ```
   ((pt1–center) * yaxis) * (pt1–center)
   ```

If pt1 equals pt2, this API creates a full circle.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_bezier

Function: Construction Geometry, Model Geometry

Action: Creates a cubic Bezier curve given four control points.

Prototype:
```
outcome api_curve_bezier (
    const SPAposition& pt1, // first control point
    const SPAposition& pt2, // second control point
    const SPAposition& pt3, // third control point
    const SPAposition& pt4, // last control point
    EDGE*& crv,             // created Bezier
                            // curve returned
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API creates an edge defined by a cubic Bezier curve specified by four control points (pt1, pt2, pt3, and pt4). The Bezier curve is created inside the control points. The curve is represented internally as a spline curve.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_ellipse

Function: Construction Geometry, Model Geometry

Action: Obsolete: Use api_mk_ed_ellipse instead.
Creates an ellipse parallel to the xy-plane of the active working coordinate system.

Prototype:
```
outcome api_curve_ellipse (
    const SPAposition& center,// center of the
                              // ellipse
    const SPAposition& major,// position on ellipse
                              // on
                              // major axis
    double ratio,             // ratio between major
                              // and minor axis length
    double start_angle,       // start angle in radians
    double end_angle,         // end angle in radians
    EDGE*& ell,               // created ellipse
                              // returned
    AcisOptions* ao = NULL    // acis options
    );
```

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

Description: This API creates an edge that represents a bounded elliptical curve with the specified center, position on major axis (pt1), and ratio. By default, the direction of the edge is always in the direction indicated by the right-hand rule relative to the normal. To create a full or partial ellipse, specify start_angle and end_angle.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_fillet

Function: Construction Geometry, Model Geometry

Action: Creates a fillet between two curves.

Prototype:
```
outcome api_curve_fillet (
    const entity_with_ray& crv1,// first curve plus
                                // approximate
                                // tangent location
    const entity_with_ray& crv2,// second curve plus
                                // approximate
                                // tangent location
    double radius,              // fillet radius
    logical trim1,              // flag to trim first
                                // curve
    logical trim2,              // flag to trim
                                // second curve
    EDGE*& arc,                 // created fillet
                                // returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/geomhusk/entwray.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates an edge defined by a circular arc of a given radius tangent to two curves (crv1 and crv2). Each curve may optionally be trimmed at the tangent point. The selection rays determine the portion of the entities that remain. The location, or ray, relates to a specific implicit position on the entity. The system-created arc is tangent to the entities. If trim1 or trim2 are FALSE, the input curve is not trimmed. If trim1 and/or trim2 is TRUE, then the corresponding input curve is trimmed to the fillet. Only fillets less than or equal to 180 degrees are created.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_law

Function: Construction Geometry, Model Geometry, Laws

Action: Creates a curve from a law.

Prototype:
```
outcome api_curve_law (
    law* in_law,            // defining law
    double start,           // start of edge
    double end,             // end of edge
    curve*& new_curve,      // curve created
    int law_number          // for future use
        = 0,
    law** other_laws        // for future use
        = NULL,
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerngeom/curve/curdef.hxx"
#include "lawutil/law_base.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates and returns a curve created from a law that takes one value and returns three values.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_line

Function: Construction Geometry, Model Geometry

Action: Creates a line through two positions.

Prototype:
```
outcome api_curve_line (
    const SPAposition& pt1, // start of line
    const SPAposition& pt2, // end of line
    EDGE*& line,            // end of line returned
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API creates and returns a straight edge with a start VERTEX and an end VERTEX defined by pt1 and pt2.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_line_tangent

Function: Model Geometry, Construction Geometry

Action: Creates a line tangent to two curves or through a position and tangent to a curve.

Prototype:
```
outcome api_curve_line_tangent (
    const SPAposition* pt1,          // start position
                                     // or NULL
    const entity_with_ray* eray1,    // first curve
                                     // plus tangent
                                     // guess or NULL
    const SPAposition* pt2,          // end position
                                     // or NULL
    const entity_with_ray* eray2,    // second curve
                                     // plus tangent
                                     // guess or NULL
    EDGE*& line,                     // created line
                                     // returned
    AcisOptions* ao = NULL           // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/geomhusk/entwray.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/edge.hxx″
#include ″baseutil/vector/position.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: In this API, either pt1 or eray1 is NULL, and either pt2 or eray2 is NULL. If both pt1 and pt2 are non-NULL, this API creates a line between the two positions. If one of the positions is non-NULL and the other is NULL, this API creates a line from the given position and tangent to the given curve. If both positions are NULL, and both entity_with_rays are non-NULL, this API creates a line tangent to the two curves.

The ray portion of the entity_with_ray determines the tangent point to use if there is more than one and also determines the start point for computing the tangent point.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_spline

Function: Construction Geometry

Action: Creates a spline curve that interpolates an array of positions.

Prototype:

```
outcome api_curve_spline (
    int numpts,                 // number of
                                // positions
    const SPAposition* pts,     // array of positions
    const SPAunit_vector* start,// direction at start
                                // or NULL
    const SPAunit_vector* end,  // direction at end
                                // or NULL
    EDGE*& crv,                 // created spline
                                // returned
    logical approx_ok = TRUE,   // TRUE if approx
                                // is ok
    logical periodic            // make periodic
        = FALSE,                // flag
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/unitvec.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates an edge that interpolates a series of points (numpts). The start and end vectors specify the direction at the ends of the curve (crv). The spline passes through the specified positions in the list (pts).

If approx_ok flag is TRUE (default), then the curve will be within SPAresfit of the points. Set approx_ok to FALSE to force the curve through each point as tightly as possible.

If the periodic flag is set to TRUE, then the curve will be made periodic, as long as the first point is within SPAresabs of the last point (i.e. the curve is closed) so that no end conditions are imposed.

If either start–direction or end–direction is not specified, a free end condition is used at that end. A "free" or "natural" spline condition means that the second derivative at that end is 0.

Errors: None

Limitations: numpts must be at least 2.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_curve_spline2

Function: Construction Geometry

Action: Creates a spline curve that interpolates an array of positions at specified parameter values.

Prototype:
```
outcome api_curve_spline2 (
    int numpts,              //number of positions
    const SPAposition* pts,  //array of positions
    const double* params,    //array of parameters
                             //values (1 for each
                             //position)
    const SPAvector* start,  //derivative at start
                             //or NULL
    const SPAvector* end,    //derivative at end
                             //or NULL
    EDGE*& crv,              //created spline returned
    AcisOptions* ao          //ACIS options such as
        = NULL               //version and journal
    );
```

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

Description: This API creates an edge that interpolates a series of points (numpts). The start and end vectors specify the derivative of curve (crv) at its ends. The spline passes through the specified positions in the list (pts) at the specified parameter values (params).
If either start-direction or end-direction is not specified, a free end condition is used at that end. A "free" or "natural" spline condition means that the second derivative at that end is 0.

Errors: None

Limitations: The number of points specified must be at least 2. Parameter values and positions should be distinct. Parameter values must be monotonic.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_edge

Function: Construction Geometry, Model Topology

Action: Creates a new edge which is a copy of the specified edge.

Prototype:
```
outcome api_edge (
    EDGE* edgeIn,           // edge to copy
    EDGE*& edgeOut,         // new edge returned
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API creates a new EDGE pointing to a new start VERTEX, a new end VERTEX, a new CURVE, and a copy of the geometry of the input edge. If the original EDGE is part of a BODY with a TRANSFORM, the elements of the new EDGE are transformed.

Errors: Pointer to edge is NULL, pointer is not to an EDGE, or EDGE is not part of a BODY.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_edge_arclength_metric

Function: Construction Geometry, Model Topology

Action: Creates a arclength parameterized curve from a curve.

Prototype:
```
outcome api_edge_arclength_metric (
    EDGE* edgeIn,           // edge to copy
    double& metric,         // new edge returned
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API creates a new EDGE pointing to a new start VERTEX, a new end VERTEX, a new CURVE, and a copy of the geometry of the input edge. If the original EDGE is part of a BODY with a TRANSFORM, the elements of the new EDGE are transformed.

A metric of zero indicates 100% arc length parameterized. A metric over one may give ACIS some problems. A metric over 100 is typically a bad curve.

Errors: Pointer to edge is NULL, pointer is not to an EDGE, or EDGE is not part of a BODY.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_edge_arclength_param

Function: Construction Geometry, Model Topology

Action: Creates a new arcwise edge which is a copy of the specified edge.

Prototype:
```
outcome api_edge_arclength_param (
    EDGE* edgeIn,           // edge to copy
    logical approx_ok,      // approximation ok
    double tol,             // tolerance
    EDGE*& edgeOut,         // arc length edge
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates a new EDGE pointing to a new start VERTEX, a new end VERTEX, and an arcwise form of the geometry of the input edge is placed on the output EDGE.

Errors: Pointer to edge is NULL, pointer is not to an EDGE, or EDGE is not part of a BODY.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_edge_law

Function: Construction Geometry, Model Geometry, Laws

Action: Creates an edge from a law.

Prototype:
```
outcome api_edge_law (
    law* inlaw,              // defining law
    double start,            // start of edge
    double end,              // end of edge
    EDGE*& new_edge,         // created edge
    int law_number           // for future use
        = 0,
    law** other_laws         // for future use
        = NULL,
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "lawutil/law_base.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates and returns an edge created from a law that takes one value and returns three values.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_edge_plaw

Function: Construction Geometry, Model Geometry, Laws

Action: Creates an edge from a R2 law and a face.

Prototype:
```
outcome api_edge_plaw (
    FACE* in_face,          // underlying face
    law* inlaw,             // defining law
    double start,           // start of edge
    double end,             // end of edge
    EDGE*& new_edge,        // created edge
    int law_number          // for future use
        = 0,
    law** other_laws        // for future use
        = NULL,
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "lawutil/law_base.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates and returns an edge created from a law that takes one value and returns two values in surface parameter space.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_edge_spiral

Function: Construction Geometry, Model Geometry, Model Object

Action: Creates an edge from a spiral definition.

Prototype:

```
outcome api_edge_spiral (
    SPAposition& center,     // start of normal axis
    SPAvector& normal,       // normal axis
    SPAposition&             // starting position
        start_position,      // of spiral
    double width,            // distance between loops
    double angle,            // radians turned
    EDGE*& spiral,           // edge returned
    logical handedness       // right or left
        = TRUE,              // handed
    AcisOptions* ao = NULL   // acis options
    );

outcome api_edge_spiral (
    SPAposition& center,     // start of normal axis
    SPAvector& normal,       // normal axis
    SPAvector& start_direction,// spiral start
                             // direction
    double width,            // distance between loops
    double angle,            // radians turned
    EDGE*& spiral,           // edge returned
    logical handedness       // right or left
        = TRUE,              // handed
    double start_radius      // length out start
        = –1.0,              // direction
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”constrct/kernapi/api/cstrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/edge.hxx”
#include ”baseutil/logical.h”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/vector.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API creates an Archimedian spiral edge. This implies that the radius expands at a constant rate. The resulting edge is perpendicular to the normal axis specified by the center and normal arguments.

The edge returned has information associated with it that indicates where the axis of the spiral is located. If the edge is then swept, a rail law is created that orients the profile relative to the center axis, rather than using a minimum rotation rail. This improves the speed of sweeping, and results in the intuitive sweeping of a spiral.

The API is overloaded to accept the start of the spiral as either a start position or a start direction. The start_position is the position where the spiral will start. The start_direction is a vector specifying an offset from the center. It should be noted that the resulting edge will be perpendicular to the axis, but will not necessarily be in the plane of the center position, depending on the starting position or direction specified.

The rate at which the spiral expands is specified with the width argument. The width is the perpendicular distance between consecutive loops of the spiral. The angle argument specifies how many revolutions the spiral should make, in radians.

The optional argument handedness determines whether the direction of movement is right handed or left handed with respect to the normal vector. The default of TRUE is right handed.

The optional argument start–radius allows a specification of the start point at a desired distance in the start direction. One use for this argument would be to specify a zero start radius so that the spiral begins at the center.

Refer to *Appendix D, Spring Scheme Examples*, for additional illustrations and Scheme examples.

Errors: Specifying a starting position or direction on the normal axis.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_edge_spring

Function: Construction Geometry, Model Geometry, Model Object

Action: Creates an edge from a spring definition.

Prototype:

```
outcome api_edge_spring (
    SPAposition& axis_point,        // axis start
    SPAvector& axis_vector,         // axis direction
    SPAposition& start_position,    // spring start
                                    // position
    logical handedness,             // right or left
    int helix_count,                // number of
                                    // helical
                                    // sections
    double*                         // pitch
        thread_distance_array,      // array
    double*                         // radians for
        rotation_angle_array,       // helical
                                    // section
    double*                         // height of
        transition_height_array,    // transition
    double*                         // radians of
        transition_angle_array,     // transition
    EDGE*& crv,                     // result spring
    AcisOptions* ao = NULL          // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "baseutil/logical.h"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/vector.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description:

This API strings together several helix pieces into a “spring” edge. Transition pieces between the helix pieces ensure G2 continuity. The final edge consists of a helix followed by zero or more transition/helix pairs. The resulting “spring” edge is returned in crv.

The edge returned has information associated with it that indicates where the axis of the helix is located. If the edge is then swept, a rail law is created that orients the profile relative to the center axis, rather than using a minimum rotation rail. This improves the speed of sweeping, and results in the intuitive sweeping of a spring.

The axis_point, axis_vector, and start_position arguments determine the location of the multi-helix. The radius of the helix is calculated as the distance from the start position to the axis.

The handedness argument determines the spring direction with respect to the axis vector. TRUE is right handed, and FALSE is left handed.

The rotation_angle_array argument determines how far the given spring section rotates about the axis and can be greater than one revolution (2 pi radians). The thread_distance_array argument determines the distance between adjacent loops, assuming that the rotation is greater than one revolution.

The transition_height_array argument determines the height distance for a transition piece, while transition_angle_array dictates the rotation (in radians) that occurs over the transition piece.

The specification of a transition region and second spring area is optional. The parameters are supplied in arrays so that any number of additional transition regions and additional api_edge_spring can be specified as long as the set of all four parameters is given for each section.

Refer to *Appendix D, Spring Scheme Examples*, for additional illustrations and Scheme examples.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_edge_spring_law

Function: Construction Geometry, Model Geometry, Model Object, Laws

Action: Creates an edge from a spring definition and a radius law.

Prototype:

```
outcome api_edge_spring_law (
    SPAposition& axis_point,          // axis point
    SPAvector& axis_vector,           // axis direction
    SPAposition& start_position,      // start position
    law* variable_radius_law,         // law specifying
                                      // spring radius
    logical handedness,               // right or left
    int helix_count,                  // thread dist
    double*                           // array thread
        thread_distance_array,        // distance
    double*                           // array of
        rotation_angle_array,         // rotation angle
    double*                           // array heights
        transition_height_array,      // for transition
    double*                           // array angles
        transition_angle_array,       // for transition
    EDGE*& crv,                       // result spring
    AcisOptions* ao = NULL            // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "baseutil/logical.h"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/vector.hxx"
#include "lawutil/law_base.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description:

This API strings together several helix pieces into a “spring” edge. Transition pieces between the helix pieces ensure G2 continuity. This API differs from api_edge_spring in that a law can be used to specify the radius of the helix. For example, a spring that expands and contracts can be created. Like api_edge_spring, the final edge consists of a helix followed by zero or more transition/helix pairs. The resulting “spring” edge is returned in crv.

The edge returned has information associated with it that indicates where the axis of the helix is located. If the edge is then swept, a rail law is created that orients the profile relative to the center axis, rather than using a minimum rotation rail. This improves the speed of sweeping, and results in the intuitive sweeping of a spring.

The axis_point, axis_vector, and start_position arguments determine the location of the multi-helix. The radius of the helix is calculated as the distance from the start position to the axis.

The radius_law argument allows specification of the radius as a function of the height of the spring. While this extension accepts a start position, the actual spring may start in a different position, depending on the starting value of the law function.

The handedness argument determines the spring direction with respect to the axis vector. TRUE is right handed, and FALSE is left handed.

The rotation_angle_array argument determines how far the given spring section rotates about the axis and can be greater than one revolution (2 pi radians). The thread_distance_array argument determines the distance between adjacent loops, assuming that the rotation is greater than one revolution.

The transition_height_array argument determines the distance for a transition piece, while transition_angle_array dictates the rotation (in radians) that occurs over the transition piece.

The specification of a transition region and second spring area is optional. The parameters are supplied in arrays so that any number of additional transition regions and additional spring areas can be specified as long as the set of all four parameters is given for each section.

Refer to *Appendix D, Spring Scheme Examples*, for additional illustrations and Scheme examples.

Errors: An incorrect law could result in a self-intersecting edge.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_edge_spring_taper

Function: Construction Geometry, Model Geometry, Model Object

Action: Creates an edge from a tapered spring definition.

Prototype:

```
outcome api_edge_spring_taper (
    SPAposition& axis_point,          // axis point
    SPAvector& axis_vector,           // axis direction
    SPAposition& start_position,      // start position
    double taper_angle,               // radius angle
                                      // in radians
    logical handedness,               // right or left
    int helix_count,                  // thread dist
    double*                           // array thread
        thread_distance_array,        // distance
    double*                           // array of
        rotation_angle_array,         // rotation angle
    double*                           // array heights
        transition_height_array,      // for transition
    double*                           // array angles
        transition_angle_array,       // for transition
    EDGE*& crv,                       // result spring
    AcisOptions* ao = NULL            // acis options
    );
```

Includes:

```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/edge.hxx″
#include ″baseutil/logical.h″
#include ″baseutil/vector/position.hxx″
#include ″baseutil/vector/vector.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description:

This API strings together several helix pieces into a “spring” edge. Transition pieces between the helix pieces ensure G2 continuity. This API differs from api_edge_spring in that an angle can be used to specify the taper of the helix. For example, a conical spring can be created. Like api_edge_spring, the final edge consists of a helix followed by zero or more transition/helix pairs. The resulting “spring” edge is returned in crv.

The edge returned has information associated with it that indicates where the axis of the helix is located. If the edge is then swept, a rail law is created that orients the profile relative to the center axis, rather than using a minimum rotation rail. This improves the speed of sweeping, and results in the intuitive sweeping of a spring.

The axis_point, axis_vector, and start_position arguments determine the location of the multi-helix. The radius of the helix is calculated as the distance from the start position to the axis.

The taper_angle argument gives the angle by which the spring radius changes in relation to the height of the spring. The radius increases for positive angles and decreases for negative angles.

The handedness argument determines the spring direction with respect to the axis vector. TRUE is right handed, and FALSE is left handed.

The rotation_angle_array argument determines how far the given spring section rotates about the axis and can be greater than one revolution (2 pi radians). The thread_distance_array argument determines the distance between adjacent loops, assuming that the rotation is greater than one revolution.

The transition_height_array argument determines the distance for a transition piece, while transition_angle_array dictates the rotation (in radians) that occurs over the transition piece.

The specification of a transition region and second spring area is optional. The parameters are supplied in arrays so that any number of additional transition regions and additional spring areas can be specified as long as the set of all four parameters is given for each section.

Refer to *Appendix D, Spring Scheme Examples*, for additional illustrations and Scheme examples.

Errors: A negative taper_angle could result in a self-intersecting edge.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_edge_to_spline

Function: Construction Geometry, Model Geometry, Model Object

Action: Creates a spline edge using a given edge.

Prototype:

```
outcome api_edge_to_spline (
    EDGE* e1,                // given edge
    EDGE*& e2,               // new spline edge
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: This API creates an edge whose underlying geometry is a spline curve (i.e. an intcurve) using the geometry underlying the given edge. It first creates an intcurve from the geometry of the given edge. It then uses the new intcurve to create a new spline edge. The direction of the new edge is the same as given edge. If the argument for the new spline edge is a NULL reference, the existing edge is converted to a spline edge.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_ed_inters_to_ents

Function: Construction Geometry, Booleans, Intersectors

Action: Creates vertex and edge entities corresponding to a list of edge-edge intersections as produced by api_inter_ed_ed.

Prototype:
```
outcome api_ed_inters_to_ents (
    EDGE* e1,                // first edge given to
                             // inter_ed_ed
    curve_curve_int* inters, // list of intersection
                             // information from
                             // inter_ed_ed
    ENTITY_LIST& ents,       // returned vertices and
                             // edges corresponding to
                             // intersections
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "kernel/kernint/intcucu/intcucu.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Refer to Action.

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

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_enclose_void

Function: Construction Geometry, Model Topology, Model Object

Action: Modifies an enclosing set of faces from void bounding to material bounding.

Prototype:
```
outcome api_enclose_void (
    FACE* face,              // center face
    REVBIT sense,            // inside/outside sense
    logical lumps,           // search for lumps if
                             // TRUE
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "kernel/meshhusk/surface/mshdef.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API walks outward from the face along the closest radial enclosing faces until a volume is enclosed or NULL coedges are encountered. Changes the containment data on the faces that are detected so they no longer contain a void. An OUTSIDE face is changed to single-sided (unless it is detected twice in the search) and a single-sided face is changed to INSIDE. If the lumps argument is TRUE, search for lumps that contain or are contained by the face group detected, and change the face containment information to reflect a filling of the void. Does not change any lumps that are not directly contained or containing; i.e., are walled-off from the void, containment-wise, by another lump. Merges any changed lumps into the original lump. Requires no action if the side of the face given is already a material containing.

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

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_ent_area

Function: Construction Geometry, Physical Properties

Action: Determines the area of a face, shell, lump, or body.

Prototype:
```
outcome api_ent_area (
    ENTITY* ent,                    // face, shell,
                                    // lump, or body
    double req_rel_accy,            // requested
                                    // relative
                                    // accuracy
    double& area,                   // returned area
                                    // of the entity
    double& est_rel_accy_achieved,  // estimate of
                                    // relative
                                    // accuracy
                                    // returned
    AcisOptions* ao = NULL          // acis options
    );
```

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

Description: Calculates the area of a face, shell, lump, or body. The req_rel_accy is passed as a fractional number greater than zero.

Errors: The pointer to the entity is NULL or does not point to a body, lump, shell, or face. Negative accuracy requested.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Read-only

api_face_conic

Function: Construction Geometry, Model Geometry

Action: Creates a conical face.

Prototype:
```
outcome api_face_conic (
    double radius,          // vertex radius
    double conic_const,     // conic constant
    double extent,          // x^2+y^2<extent or
                            // y^2<extent if length>0
    double length,          // if length>0 then a
                            // conic trough is formed
    FACE*& face,            // output face returned
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: Refer to Action.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_face_cylinder_cone

Function: Construction Geometry, Model Geometry

Action: Creates a cylindrical or conical face.

Prototype:
```
outcome api_face_cylinder_cone (
    const SPAposition& center,  // axis origin
    const SPAvector& normal, // axis direction and
                             // height
    double bottom,           // bottom radius
    double top,              // top radius
    double start,            // start angle
    double end,              // end angle
    double ratio,            // elliptical
                             // major/minor ratio
    const SPAposition* pt,   // point defining major
                             // axis
    FACE*& face,             // created face returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/vector.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates a cylindrical or conical face, given a center, an axis direction and height (normal), a top radius, and a bottom radius. The face start angle (start) and end angle (end) derive from the active WCS or, if given, from a coordinate system derived from the axis (normal) and a point (pt) on the major axis. All positions are relative to the active WCS.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_face_law

Function: Construction Geometry, Model Geometry, Laws

Action: Creates a law face.

Prototype:
```
outcome api_face_law (
    law* in_law,              // defining law
    double minu,              // face min boundary u
    double maxu,              // face max boundary u
    double minv,              // face min boundary v
    double maxv,              // face max boundary v
    FACE*& face,              // face created
    int in_law_number         // for future use
        = 0,
    law** in_other_laws       // for future use
        = NULL,
    AcisOptions* ao = NULL    // acis options
    );
```

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

Description: The law must map two dimensional parameter space (*u*,*v*) to three dimensional object space (*x*, *y*, *z*). The min and max values specify face boundaries in parameter space.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_face_plane

Function: Construction Geometry, Model Geometry

Action: Creates a planar face.

Prototype:
```
outcome api_face_plane (
    const SPAposition& p,    // anchor point
    double width,            // face width
    double height,           // face height
    const SPAvector* normal, // plane normal
    FACE*& face,             // created face returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/face.hxx″
#include ″baseutil/vector/position.hxx″
#include ″baseutil/vector/vector.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API creates a planar face. All positions are relative to the active WCS or, if a normal is specified, a temporary coordinate system derives from the normal and the *x*-axis.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_face_sphere

Function: Construction Geometry, Model Geometry

Action: Creates a spherical face.

Prototype:
```
outcome api_face_sphere (
    const SPAposition& center,  // center
    double radius,              // radius
    double lo_start,            // longitudinal start
                                // angle
    double lo_end,              // longitudinal end angle
    double la_start,            // latitudinal start
                                // angle
    double la_end,              // latitudinal end angle
    const SPAvector* normal,    // pole direction
    FACE*& face,                // created face returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/face.hxx″
#include ″baseutil/vector/position.hxx″
#include ″baseutil/vector/vector.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API creates a spherical face aligned with the axes of the working coordinate system. The North pole is parallel to the *y*-axis. Longitude points toward the North pole and latitude points toward the zero meridian. All positions are relative to the active WCS or, if normal is specified, a coordinate system derives from the normal and the *x*-axis.

Angle ranges:

Longitude	–90 to 90 degrees, 0 at the equator
Latitude	–360 to 360 degrees, 0 at the meridian

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_face_spl_apprx

Function: Construction Geometry, Model Geometry

Action: Creates a spline face which is fit to a grid of positions.

Prototype:
```
outcome api_face_spl_apprx (
    const splgrid* grid,     // spline surface grid
    FACE*& face,             // created face returned
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: A grid is an array of points that lie on the spline surface and whose representation has a grid-like appearance. This API creates a spline surface by approximating the points on the grid to a specified tolerance.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_face_spl_ctrlpts

Function: Construction Geometry, Model Geometry

Action: Creates a spline face from control points and knots.

Prototype:
```
outcome api_face_spl_ctrlpts (
    const splsurf* surf,     // spline surface data
    FACE*& face,             // created face returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”constrct/geomhusk/splsurf.hxx”
#include ”constrct/kernapi/api/cstrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API creates a spline surface face (surf). All surface data is wrapped in an object of the splsurf class.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_face_spl_intrp

Function: Construction Geometry, Model Geometry

Action: Creates a spline surface face.

Prototype:
```
outcome api_face_spl_intrp (
    const splgrid* grid,     // spline surface grid
    FACE*& face,             // created face returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/geomhusk/splgrid.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/face.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: A grid is an array of points that lie on the spline surface and whose representation has a grid-like appearance. This API creates a spline surface by interpolating the points on the grid. Initialize the grid with points on the spline surface and any tangent vectors defining the surface boundaries.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_face_torus

Function: Construction Geometry, Model Geometry

Action: Creates a toroidal face.

Prototype:
```
outcome api_face_torus (
    const SPAposition& center,  // center
    double major,           // major radius
    double minor,           // minor radius
    double tu_start,        // tube (u) start angle
    double tu_end,          // tube (u) end angle
    double sv_start,        // spine (v) start angle
    double sv_end,          // spine (v) end angle
    const SPAvector* normal,// axis direction
    FACE*& face,            // created face returned
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/face.hxx″
#include ″baseutil/vector/position.hxx″
#include ″baseutil/vector/vector.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API creates a toroidal face aligned with the axes of the working coordinate system. All positions are relative to the active WCS or, if normal is specified, a coordinate system derives from the normal and the *x*-axis.

Angle ranges:
tube (*u*) . . . . . = –360 to 360
spine (*v*) . . . . . = –360 to 360

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_fillet_vertex

Function: Construction Geometry, Model Geometry

Action: Fillets a wire at a given vertex.

Prototype:
```
outcome api_fillet_vertex (
    VERTEX* vert,           // vertex to fillet at
    double radius,          // radius of fillet
    EDGE* edge1             // for future use
        = NULL,             // for future use
    EDGE* edge2             // for future use
        = NULL,             // for future use
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: Refer to Action.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_find_face

Function: Construction Geometry, Object Relationships

Action: Determines a planar face with normal in given direction.

Prototype:
```
outcome api_find_face (
    BODY* body,                          // body
                                         // containing
                                         // face sought
    SPAunit_vector const& direction,  // approximate
                                         // direction of
                                         // face normal
    FACE*& face,                         // found face
                                         // returned or
                                         // NULL returned
                                         // if no face
                                         // found
    AcisOptions* ao = NULL               // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/body.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "baseutil/vector/unitvec.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Given a direction *d*, finds planar face with normal *n* so that $d \cdot n$ is positive and has greatest value for any face of the body. If several faces have equal greatest value of $d \cdot n$, it returns the face with greatest value of $p \cdot d$, where *p* is a point on the face. Takes account of the body transformation.

Retrieves NULL if no face found with $d \cdot n$ positive.

Errors: Pointer to body is NULL or not to a BODY.
Zero length direction vector specified.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Read-only

api_find_vertex

Function: Construction Geometry, Object Relationships

Action: Gets the vertex closest to given position.

Prototype:
```
outcome api_find_vertex (
    BODY* body,              // body containing vertex
                             // sought
    SPAposition const& pos,  // given position in
                             // global coordinates
    VERTEX*& vertex,         // found vertex returned
                             // or NULL returned
                             // if no vertex found
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/body.hxx"
#include "kernel/kerndata/top/vertex.hxx"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Given a position and a body, finds the vertex of the body closest to that position. If several vertices are equidistant from the given position, any one may be returned.

Returns NULL if no vertex found.

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

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Read-only

api_initialize_constructors

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

Action: Initializes the constructor library.

Prototype:
```
outcome api_initialize_constructors ();
```

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

Description: Refer to Action.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: System routine

api_loop_external

Function: Construction Geometry, Model Topology

Action: Determines if a loop is internal or external.

Prototype:
```
outcome api_loop_external (
    LOOP* given_loop,        // loop to be examined
    logical* if_external,    // TRUE returned if
                             // external
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/loop.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Refer to Action.

Errors: Pointer to loop is NULL or not to a LOOP.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Read-only

api_make_approx_curve

Function: Spline Interface, Construction Geometry

Action: Creates a curve that approximates a given curve.

Prototype:
```
outcome api_make_approx_curve (
    curve const* input_curve,   // given curve
    double const tol,           // tolerance for
                                // approximation
    SPAinterval range,          // range for approx.
                                // curve
    curve*& output_curve,       // approximating
                                // curve returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerngeom/curve/curdef.hxx″
#include ″baseutil/vector/interval.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: Creates a curve that approximates a given curve to a specified tolerance over the specified range, which must be contained within the range of the given curve. This function is intended for applications performing translation and needing an approximation of the curve.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_approx_surface

Function: Spline Interface, Construction Geometry

Action: Creates a surface that approximates a given surface.

Prototype:

```
outcome api_make_approx_surface (
    surface const* input_surface,   // given surface
    double const tol,               // tolerance for
                                    // approximation
    SPAinterval const& u_range,
// u range for
                                    // returned
                                    // surface
    SPAinterval const& v_range,
// v range for
                                    // returned
                                    // surface
    surface*& output_surface,       // resulting
                                    // approximating
                                    // surface
    AcisOptions* ao = NULL          // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerngeom/surface/surdef.hxx"
#include "baseutil/vector/interval.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Creates a surface that approximates a given surface to a given tolerance over the specified ranges, which must be contained within the range of the given surface. This function is intended for applications performing translation and needing an approximation of the surface.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_cnface

Function: Construction Geometry, Model Geometry

Action: Creates a face that is a portion of a cone.

Prototype:

```
outcome api_make_cnface (
    const SPAposition& center,      // center of the
                                    // base ellipse
    const SPAunit_vector& normal_axis,// normal to
                                    // base
                                    // ellipse
    const SPAvector& major_axis,    // major axis of
                                    // base ellipse
    double radius_ratio,            // ratio of the
                                    // major to minor
                                    // axes
    double sint,                    // sine of the
                                    // half angle of
                                    // the cone
    double cost,                    // cosine of the
                                    // half angle of
                                    // the cone
    double st_ang,                  // start angle
                                    // (radians) of
                                    // cone
    double end_ang,                 // end angle
                                    // (radians) of
                                    // cone
    double height,                  // height of cone
    FACE*& face,                    // created
                                    // conical face
                                    // returned
    AcisOptions* ao = NULL          // acis options
    );
```

Includes:

```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/face.hxx″
#include ″baseutil/vector/position.hxx″
#include ″baseutil/vector/unitvec.hxx″
#include ″baseutil/vector/vector.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: The cone is defined by a base ellipse, sine, and cosine of the coning angle.

Specify the base ellipse by a center point, a unit vector normal to the plane of the ellipse, the major axis vector, and ratio of the minor to major axis length.

The sine and cosine of the coning angle specify whether the result is a cylinder or a TRUE cone. If it is a cone, they specify the direction of the apex.

The boundaries of the conical face are specified by start_ang, end_ang, and height. The angles are in the range [–2 pi, 2 pi].

The end of the major axis is 0 and it increases in compliance to the right hand rule using the ellipse normal.

The height can be positive or negative, with the plane of the ellipse being 0, and positive numbers in the direction of the ellipse normal.

The cone starts at the plane of the ellipse. It has the specified height, but never extends past the apex of the cone.

Errors: Zero length normal or major axis vector specified.
Major axis not perpendicular to normal.
Zero height specified.
Radius ratio not greater than zero.
Start and end angles equal.
Sine and cosine of half angle both zero.

Limitations: The difference (end – start) is limited to between 0 and 2 pi.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_cuboid

Function: Construction Geometry, Model Geometry, Model Object

Action: Creates cuboid of given width, depth and height.

Prototype:
```
outcome api_make_cuboid (
    double width,           // size in x
    double depth,           // size in y
    double height,          // size in z
    BODY*& body,            // cuboid returned
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API constructs a cuboid centered at the origin. All arguments must be greater than SPAresabs.

Errors: Width, depth, or height not greater than SPAresabs.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_ewire

Function: Construction Geometry, Model Geometry

Action: Creates a body that consists a single wire from a list of edges that are connected in the order given with no branching.

Prototype:
```
outcome api_make_ewire (
    int num_edges,          // number of edges
                            // in list
    EDGE* edges[],          // list of edges
    BODY*& body,            // created wire body
                            // returned
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/body.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates a BODY that consists of a single WIRE. The COEDGEs are created from the EDGEs in the edge list of the wire. Duplicate VERTEXs between adjacent edges are deleted. The edges are assumed to form a C1 continuous curve that is not self-intersecting. The edges may be open or closed.

A single point wire can be made. However, such wires are not fully supported by Booleans and offsetting.

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

Limitations: Wires that are in single isolated point (characterized by an edge with null geometry) are not fully supported.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_ewires

Function: Construction Geometry, Model Geometry

Action: Separates and sorts a possibly branched list of edges into a group of non-branched wire bodies.

Prototype:
```
outcome api_make_ewires (
    int num_edges,          // edge count
    EDGE* edges[],          // list of edges
    int& n_bodies,          // number of wire
                            // bodies created
    BODY**& bodies,         // wire bodies created
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/body.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Input edges need not be ordered.

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

Limitations: Wires that are in a single isolated point (characterized by an edge with null geometry) are not fully supported.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_edge_from_curve

Function: Construction Geometry, Model Object, Model Topology

Action: Creates an edge using a copy of the input curve.

Prototype:
```
outcome api_make_edge_from_curve (
    curve const* cur,        // given curve
    EDGE*& edge,             // returned edge
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "kernel/kerngeom/curve/curdef.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This is a convenient way to create model topology (e.g., an edge) from construction geometry (e.g., a curve).

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_frustum

Function: Construction Geometry, Model Geometry, Model Object

Action: Creates an elliptical cone or cylinder of given height and radii.

Prototype:
```
outcome api_make_frustum (
    double height,           // height
    double rad1,             // radius in x-direction
                             // at base
    double rad2,             // radius in y-direction
                             // at base
    double top,              // radius in x-direction
                             // at top
    BODY*& body,             // created frustum
                             // returned
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: This API creates an elliptical cone or cylinder of given height and radii.

Centers the frustum at the origin, with main axis along the *z*-axis and one axis (major or minor) of its cross-section in the $z = 0$ plane along the *x*-axis.

All arguments must be greater than SPAresabs.

Errors: Either height, rad1, or rad2 is not greater than SPAresabs or top is not greater than or equal to 0.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_kwire

Function: Construction Geometry, Model Geometry

Action: Creates a kwire body from an array of positions and bulges.

Prototype:
```
outcome api_make_kwire (
    BODY* given_body,           // existing body to
                                // add to, or NULL to
                                // create a new body
    SPAunit_vector const& normal, // normal to plane
                                // of
                                // wire
    int length_pts,             // number of span
                                // positions in array
    SPAposition array_pts[],    // array of positions
    double array_bulges[],      // array of bulge
                                // factors
    BODY*& body,                // body containing
                                // wire returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/body.hxx"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/unitvec.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API constructs an open or closed kwire wire body made up of straight or circular arc edges from an array of positions and bulge factors (the bulge of a circular arc is the tangent of a quarter of the angle subtended by the arc at its center).

Every arc lies in a plane with the given normal.

Bulge factors are positive for arcs directed counterclockwise with respect to the normal.

An array of *n* positions requires an array of *n*–1 bulge factors.

The wire is closed if the first and last points coincide within SPAresabs.

Rejects open wires of less than 2 positions and closed wires of less than 3 positions.

Determines that the chord of each segment is perpendicular to the normal, and that no segment has chord length less than SPAresabs.

Does not check that points other than first and last are distinct, or that segments do not intersect one another.

If given body is NULL, makes a new body.

Leaves new wire as first in wire list of body, new or old. Returns the body.

Errors: Pointer to given body is not to a BODY.
Zero length normal vector.
(refer to the Description for other errors)

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_planar_disk

Function: Construction Geometry, Model Geometry

Action: Creates a face that is a planar disk.

Prototype:
```
outcome api_make_planar_disk (
    const SPAposition& origin,  // origin point
    const SPAunit_vector& normal, // the normal
    double radius,              // the radius
    FACE*& face,                // output returned
    logical half_space          // If set to TRUE, a
        = FALSE,                // full half space is
                                // returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/unitvec.hxx"
#include "baseutil/logical.h"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_plface

Function: Construction Geometry, Model Geometry

Action: Creates a face that is a parallelogram specified by three points: origin, left, and right.

Prototype:
```
outcome api_make_plface (
    const SPAposition& origin,// origin of the plane
    const SPAposition& left,// left point on the
                            // plane
    const SPAposition& right,// right point on the
                            // plane
    FACE*& face,            // created plface
                            // returned
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Creates a face that is a parallelogram specified by three points: origin, left, and right. These points must not be colinear. The normal to the plane is defined by the cross product of the vectors (right – origin) and (left – origin). The boundary consists of four linear edges starting at the origin and going to the right to define a parallelogram ending at the origin.

The origin is the root point of the plane and the normal is in the direction of (right – origin) * (left – origin). Left and right are points to the left and right of the origin.

Errors: A plane cannot have a zero length or width. This means that both (right – origin) and (left – origin) must be greater than SPAresabs.

A plane cannot have a zero length normal. This means that the length of ((right – origin) * (left – origin)) must be greater than SPAresabs.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_polygon

Function: Construction Geometry, Model Geometry, Model Object

Action: Creates a polygonal wire from in a plane orthogonal to a given normal.

Prototype:

```
outcome api_make_polygon (
    BODY*& polygon,          // polygon made
    SPAposition centre,      // center of the polygon
    SPAvector start,         // from center to a
                             // vertex or the center
                             // of an edge of polygon
    SPAvector& normal,       // a vector normal to the
                             // plane of the polygon
    double& side_length,     // length of the side
                             // (this will scale
                             // ”start”)
    int number_of_sides      // number of sides
        = 6,
    logical oncenter         // TRUE when the start
        = FALSE,             // is a vector to the
                             // center of an edge
    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 ”constrct/kernapi/api/cstrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API constructs a polygonal wire. The length of the sum of the sides of the polygon is calculated and passed back.

Note that both the length of the vector start and the double length dictate the length of the sides of the polygon. If the user passes in length <= SPAresabs, the length of the sides of the polygon will be determined solely by the vector start and the number of sides in the polygon and the oncenter flag. In this case the user is told the computed length of sides by the reference variable length. If on the other hand the value of length passed in is > SPAresabs, the polygon will be scaled so that the sides each have the proper length.

Errors: The start vector must be non-zero.
The normal vector must be non-zero.
The start and normals must span a two dimensional space.
The number of sides must be at least 3

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_prism

Function: Construction Geometry, Model Geometry, Model Object

Action: Creates an elliptical prism of given height, radii, and number of sides.

Prototype:
```
outcome api_make_prism (
    double height,          // height of prism
    double rad1,            // first radius
    double rad2,            // second radius
    int nsides,             // number of sides
    BODY*& body,            // prism body returned
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API creates an elliptical prism of given height, radii, and number of sides. Center it at the origin, aligned the z-axis with one face perpendicular to the positive *x*-axis.

Errors: Either height, rad1, or rad2 is less than SPAresabs or nsides is less than 3.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_pyramid

Function: Construction Geometry, Model Geometry

Action: Creates an elliptical pyramid of given height, radii, and number of sides.

Prototype:
```
outcome api_make_pyramid (
    double height,          // height of pyramid
    double rad1,            // major radius of base
    double rad2,            // minor radius of base
    double top,             // major radius of top
    int nsides,             // number of sides
    BODY*& body,            // pyramid body returned
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API creates an elliptical pyramid of given height, radii, and number of sides. Center it at the origin, aligned with the *z*-axis and with one side of the base perpendicular to the positive *x*-axis. If the height is zero, the resulting body consists of just one polygonal–sided sheet face, lying in the *xy* plane.

Errors: Either rad1 or rad2 is less than SPAresabs or height is greater than zero and less than SPAresabs or top is less than –SPAresabs or nsides is less than 3.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_spface

Function: Construction Geometry, Model Geometry

Action: Creates a face that is a portion of a sphere.

Prototype:

```
outcome api_make_spface (
    const SPAposition& center,// center of the sphere
    double radius,          // radius of the sphere
    const SPAunit_vector& lat,// latitude direction
    const SPAunit_vector& lon,// longitude direction
    double slat,            // start angle and end
                            // angles
    double elat,            // in latitude direction
                            // (in radians)
    double slon,            // start angle and end
                            // angles
    double elon,            // in longitude
                            // direction
                            // (in radians)
    FACE*& face,            // spherical face
                            // returned
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:

```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/face.hxx″
#include ″baseutil/vector/position.hxx″
#include ″baseutil/vector/unitvec.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: The sphere is defined by its center and radius.

The boundaries of the spherical face lie along latitude and longitude lines.

The longitude vector points to the north pole and the latitude vector points to the 0 meridian. The unit vectors defining latitude and longitude lines must be perpendicular.

Latitude angles lie in the range [–pi, pi] from the south pole to the north pole.

Longitude angles lie in the range [–2 pi, 2 pi] with 0 being the meridian at the end of the latitude vector and increasing relative to the right hand rule using the longitude vector.

The subtended longitude angle must be less than or equal to 2 pi.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_sphere

Function: Construction Geometry, Model Geometry

Action: Creates a sphere of given radius.

Prototype:
```
outcome api_make_sphere (
    double radius,          // radius of desired
                            // sphere
    BODY*& body,            // sphere returned
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API creates a sphere of given radius centered at the origin.

Errors: Radius not greater than SPAresabs.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_spline

Function: Construction Geometry, Model Geometry, Model Object

Action: Creates a body consisting of a single face which is the whole of a given B-spline surface.

Prototype:
```
outcome api_make_spline (
    spline const& this_spline,  // spline surface
    BODY*& body,                // single-face body
                                // returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/body.hxx″
#include ″kernel/kerngeom/surface/spldef.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API constructs a face containing one loop with four edges surrounding the given surface. Returns this in a body.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_torus

Function: Construction Geometry, Model Geometry

Action: Creates a torus of given major and minor radii.

Prototype:
```
outcome api_make_torus (
    double major_radius,     // radius of spine of
                             // desired torus
    double minor_radius,     // radius of
                             // cross-section of ring
    BODY*& body,             // torus body returned
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: This API creates a torus of given radii centered at the origin.

Errors: minor_radius not greater than SPAresabs.
major_radius not greater than minus the minor_radius, or 0.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_trface

Function: Construction Geometry, Model Geometry

Action: Creates a face that is a portion of a torus.

Prototype:
```
outcome api_make_trface (
    const SPAposition& center,  // center of the
                                // torus
    const SPAunit_vector& normal, // normal axis of
the
                                // torus
    double major,               // major radius of
                                // the torus
    double minor,               // minor radius of
                                // the torus
    const SPAposition& pnt,     // point defining
                                // origin of
                                // parameterization
    double uf,                  // start angle and
                                // end angles
    double ut,                  // in u direction (in
                                // radians)
    double vf,                  // start angle and
                                // end angles
    double vt,                  // in v direction (in
                                // radians)
    FACE*& face,                // toroidal face
                                // returned
    AcisOptions* ao = NULL      // acis options
    );
```

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

Description: The torus is defined by a plane specified by a point and unit vector normal to the plane. The major radius gives the distance from the center to the spine curve lying in this plane around which a circle having the minor radius is swept to define the torus.

Three classes of tori can be specified: donut, apple, or lemon depending on the relative magnitudes of the major and minor radii. Also, the signs of the of the radii are significant in defining the shape of the tori and the orientation of the outward surface normals.

The point defining the origin of a parameterization scheme on the torus is projected onto the plane of the torus. The intersection of the torus and the line from the center towards this point that is farthest from the center define the (0, 0) of the parameterization scheme. The boundaries of the face lie on isoparametric curves specified by angles around the tube and around the spine. Also, the angles along and around the spine increase following the right hand rule.

Errors: None

Limitations: The range of angles (*vf* to *ut*) must be <= 2 pi.
Regular torus – donut: The range of (*uf* to *ut*) must be <= 2 pi.
Apples and lemons: the values of the *uf* and *ut* must be within the angle

acos(fabs(major)/fabs(minimum)) {singularity points}.

Values larger than that are trimmed to the singularity.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_make_wire

Function: Construction Geometry, Model Geometry

Action: Creates a polygonal wire from an array of positions.

Prototype:

```
outcome api_make_wire (
    BODY* given_body,        // body to add wire to,
                             // or NULL for new body
    int length_pts,          // number of span end
                             // points in array
    SPAposition array_pts[], // array of points
    BODY*& body,             // wire body returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/body.hxx″
#include ″baseutil/vector/position.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API constructs a polygonal wire from an array of positions. The wire is considered closed if the first and last points coincide.

Does not check that points other than adjacent points are distinct.

If given body is NULL, makes a new body. It leaves the new wire as first in wire list of body, new or old. Returns the body.

A single point wire can be made. However, such wires are not fully supported by Booleans and offsetting.

Errors: Pointer to given body is not to a BODY. Distance between adjacent points in list less than SPAresabs.

Limitations: Wires that are in single isolated point (characterized by an edge with null geometry) are not fully supported.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_manifold_class

Function: Construction Geometry, Object Relationships

Action: Determines the differences between all manifold shells, manifold sheets, and wire edges. Returns all nonmanifold edges and vertices.

Prototype:

```
outcome api_manifold_class (
    BODY* given_body,              // Body to examine
                                   // for manifold
                                   // pieces
    ENTITY_LIST*& mani_shells,     // Array of lists of
                                   // faces returned
                                   // (each list
                                   // represents a
                                   // manifold shell)
                                   // Terminated by an
                                   // empty list
    ENTITY_LIST*& mani_sheets,     // Array of lists of
                                   // manifold sheets
                                   // returned
                                   // Terminated by an
                                   // empty list
    ENTITY_LIST& wire_edges,       // List of wire edges
                                   // detected and
                                   // returned
    ENTITY_LIST& lamina_faces,     // List of lamina
                                   // (doubly covered)
                                   // faces in pairs
                                   // returned
    ENTITY_LIST*& edge_arcs,       // Array of lists
                                   // returned, each
                                   // containing one
                                   // nonmanifold edge
                                   // followed by all
                                   // elements connected
                                   // to it. Terminated
                                   // by an empty list
    ENTITY_LIST*& vertex_arcs,     // Array of lists
                                   // returned, each
                                   // containing one
                                   // nonmanifold
                                   // vertex followed
                                   // by all elements
                                   // connected to it
                                   // Terminated by an
                                   // empty list
    AcisOptions* ao = NULL         // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kerndata/top/body.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: Determines the differences between all manifold shells, manifold s

Definition of Nonmanifold

Formally, a nonmanifold edge has more than two faces around it, and a nonmanifold vertex has elements that can only be connected topologically through that vertex. For example, two cones meeting at their apexes, or a vertex of a block with a dangling edge. For the purposes of this API, three or more wire edges meeting at a vertex are also defined as nonmanifold.

Information Returned

The first three output arguments, (mani_shells, mani_sheets, wire_edges) return the detected manifold shells, manifold sheets and the wire edges. Wire edges are returned in an entity list. Manifold sheets and shells are returned in arrays of entity lists, one list for each shell or sheet that contains its faces. Each array is terminated by an empty list.

The next output argument, lamina_faces, returns all laminar (covered on both sides) faces. Only planar laminar faces or laminar faces that use the same geometry are detected. The entity list stores them in pairs (first two are one face, next two are the second face, etc.). Detects laminar faces of opposing and same orientation.

The next two output arguments (nm_edges, nm_vertexes) return all detected nonmanifold edges and nonmanifold vertices and all elements connected to them, whether sheets, lamina, manifold shells, or wire edges. Each list in either array begins with the nonmanifold edge or vertex, followed by the elements connected to them. The manifold shells and sheets are represented by a pointer that references the first face field to the entity list (in either return argument) that represents the shell or sheet. Before using this pointer, cast the contained face pointer to an ENTITY_LIST. The wire edges are stored in the list. Each lamina is stored as two sequential faces in the list. The arrays are terminated by empty lists.

Note that arguments mani_shells, mani_sheets, nm_edges, and nm_vertexes are allocated in the API code and must be deleted via delete at some point.

Locates all entities if connected to coedge graph, even if not in any shell or wire.

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

Limitations: This API assumes that the input body is more or less sane other than nonmanifold regions. Inner dangling sheet faces attached to a shell cause the shell to be interpreted as a sheet. Void shells that touch their peripheral shell or other void shells in a nonmanifold edge will be interpreted as part of the adjacent shell.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Read-only

api_mk_by_faces

Function: Construction Geometry, Model Topology

Action: Adds each FACE in the array that is attached to a SHELL within a LUMP to a BODY.

Prototype:
```
outcome api_mk_by_faces (
    BODY* given_body,        // body to which to add
    int num_faces,           // number of faces in
                             // array
    FACE* faces[],           // array of faces
    BODY*& body,             // body returned
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: For each FACE in an array, the API makes a new LUMP consisting of a single SHELL. The FACE is attached to the SHELL. No attempt is made to stitch the faces into larger sheets, and no check is made for mutually intersecting or self-intersecting faces.

If the given_body is NULL, a new BODY is created and returned. Otherwise, the old BODY is used and the LUMP is added to the list of lumps in the body. In either case, the BODY pointer is updated on return.

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

Errors: Pointer to body is not to a BODY.
A pointer in the face array is NULL or not to a FACE.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_mk_ed_bs3_curve

Function: Construction Geometry, Model Topology

Action: Creates an EDGE that represents an intersection curve defined by a 3D NURBS curve.

Prototype:
```
outcome api_mk_ed_bs3_curve (
    bs3_curve bs,           // curve
    EDGE*& edge,            // edge representing
                            // intersection curve
                            // returned
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "kernel/spline/bs3_crv/bs3curve.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API copies the NURBS curve to make the INTCURVE supported by the NURBS curve.

Creates an EDGE with supporting VERTEXs that represent the bounded edge.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_mk_ed_conic

Function: Construction Geometry, Model Topology

Action: Creates a spline edge defined by a conic.

Prototype:
```
outcome api_mk_ed_conic (
    const SPAposition& start,       // start point of
                                    // conic
    const SPAunit_vector& start_tan,  // tangent at
                                    // start point
    const SPAposition& end,         // end point of
                                    // conic
    const SPAunit_vector& end_tan,  // tangent at end
                                    // point
    double rho,                     // type of conic
                                    // (0 < rho <
                                    // 1.0)
    EDGE*& edge,                    // created edge
                                    // returned
    AcisOptions* ao = NULL          // acis options
    );
```

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

Description: Creates a spline edge from a conic, where rho determines the type of conic. Refer to the figure below for an illustration of rho.

start is the start point *A* for arc *AB*

start_tan is the vector *AC* tangent to arc *AB* at the start point

end is the end point *B* for arc *AB*

end_tan is the vector *BC* tangent to arc *AB* at the end point

point *D* is the midpoint of segment *AB*

point *E* is the intersection of arc *AB* with segment *CD*

rho = length *ED* / length *CD*, where:

rho < 0.5 is a (spline) ellipse

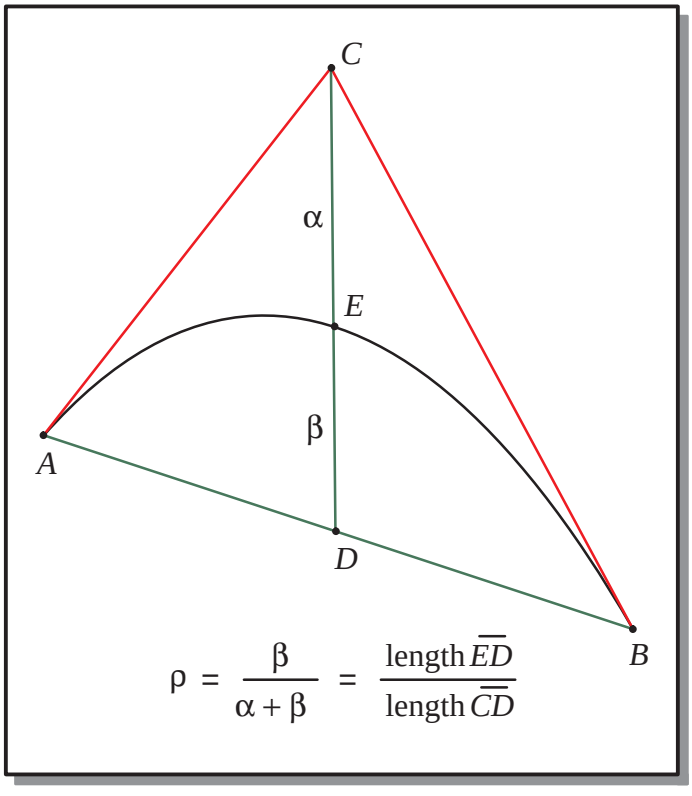
rho = 0.5 is a (spline) parabola

rho > 0.5 is a (spline) hyperbola

edge is the edge returned, such that:

ACB is the control polygon of the rational quadratic spline curve

Refer to the following figure explaining the method for determining the "rho" argument of api_mk_ed_conic.



Errors: Start/end tangents are collinear or parallel (fail to intersect)

Limitations: Cannot make ellipses whose angular extent is 180 degrees or more. The spline is always rational, even though parabolas do not need to be rational.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_mk_ed_cubic

Function: Construction Geometry, Model Topology

Action: Creates an EDGE that represents a cubic spline curve interpolating or fitting a sequence of points.

Prototype:
```
outcome api_mk_ed_cubic (
    int npts,                          // number of
                                       // points in
                                       // array
    const SPAposition pts[],           // array of
                                       // points
    SPAunit_vector const& start_dir,  // start
                                       // direction
    SPAunit_vector const& end_dir,  // end direction
    double fitol,                      // fit tolerance,
                                       // or 0 for
                                       // interpolation
    EDGE*& edge,                       // created edge
                                       // returned
    AcisOptions* ao = NULL             // acis options
    );
```

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

Description: This API creates and returns an EDGE with an INTCURVE as the geometry supporting the edge. The edge is given start and end directions.

If the fit tolerance is 0, the curve interpolates the sequence of points and has tangents at the start and end points in the direction of the unit vectors. If the fit tolerance is greater than 0, the curve is a fit to the sequence of points.

If the start and end tangent directions are identical and the start and end points are identical, the curve is marked as a periodic intcurve.

If the start and end directions are (0, 0, 0), default tangent directions are calculated using up to three points at the ends of the point array. Also, this condition can be specified by passing the NULL_unitvec found in unitvec.hxx.

No check is made to determine if the edge is self-intersecting.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_mk_ed_ellipse

Function: Construction Geometry, Model Topology

Action: Creates an EDGE that represents a bounded elliptical arc.

Prototype:

```
outcome api_mk_ed_ellipse (
    SPAposition const& center,  // center of arc
    SPAunit_vector const& normal,// normal
    SPAvector const& major_axis,// major axis of arc
    double radius_ratio,        // radius ratio
    double start_angle,         // starting angle of
                                // arc
    double end_angle,           // ending angle of
                                // arc
    EDGE*& edge,                // edge returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.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: This API creates an EDGE that represents a bounded elliptical arc.

Creates the underlying ELLIPSE with the specified center, normal, major_axis, and radius_ratio.

By default, the direction of the ELLIPSE is always in the direction indicated by the right hand rule relative to the normal.

The start and end angles (measured in radians) specify the start and end points of the elliptical arc. They are used to create vertices on the edge.

Errors: Zero length normal vector.
Zero length major axis.
Major axis not perpendicular to normal vector.
Radius ratio less than SPAresabs or greater than 1.0.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_mk_ed_int_ctrlpts

Function: Construction Geometry, Model Topology

Action: Creates an EDGE that represents an intersection curve defined by a sequence of control points and knots.

Prototype:

```
outcome api_mk_ed_int_ctrlpts (
    int degree,                    // degree
    logical rational,              // TRUE if rational
                                   // control points
    logical closed,                // TRUE if closed
                                   // curve
    logical periodic,              // TRUE if periodic
                                   // curve
    int num_ctrlpts,               // number of control
                                   // points in array
    const SPAposition ctrlpts[],// array of control
                                   // points
    const double weights[],        // array of weights
    double point_tol,              // identical points
                                   // if within this
                                   // tolerance
    int num_knots,                 // number of knots in
                                   // array
    const double knots[],          // array of knots
    double knot_tol,               // identical knots if
                                   // within this
                                   // tolerance
    EDGE*& edge,                   // edge returned
    AcisOptions* ao = NULL         // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "baseutil/logical.h"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description:

This API creates a bs3_curve of the specified degree. The bs3_curve is defined by the given sequence of control points and knots. An EDGE is created using the intcurve associated with the bs3_curve.

The parameter rational specifies whether the control points are rational or not. The coordinates of the control points are located in the array ctrlpts. If rational, the associated weights are found in the array weights.

The tolerance point_tol Determines when two control points are identical. knot_tol performs the same function for the knot sequence.

If the first and last control points are within point_tol, the curve is closed.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_mk_ed_line

Function: Construction Geometry, Model Topology

Action: Creates an EDGE that represents a bounded line segment between two points.

Prototype:
```
outcome api_mk_ed_line (
    SPAposition const& start_point,// starting point
    SPAposition const& end_point,// ending point
    EDGE*& edge,                // edge returned
    AcisOptions* ao = NULL      // acis options
    );
```

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

Description: This API creates and returns an EDGE with a start VERTEX and end VERTEX defined by given positions.

Also creates a STRAIGHT line as the geometry supporting the edge.

Errors: Distance between start and end points less than SPAresabs.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_mk_fa_spl_ctrlpts

Function: Construction Geometry, Model Topology

Action: Creates a FACE representing the spline surface defined by a sequence of control points and knots.

Prototype:

```
outcome api_mk_fa_spl_ctrlpts (
    int degree_u,                  // degree in u
    logical rational_u,            // TRUE if rational
                                   // in u
    int form_u,                    // open, closed,
                                   // periodic in u
    int pole_u,                    // singularities in u
                                   // u-minimum and/or
                                   // u-maximum
    int num_ctrlpts_u,             // number of u
                                   // control points in
                                   // array
    int degree_v,                  // degree in v
    logical rational_v,            // TRUE if rational
                                   // in v
    int form_v,                    // open, closed,
                                   // periodic in v
    int pole_v,                    // singularity at
                                   // v-minimum and/or
                                   // v-maximum
    int num_ctrlpts_v,             // number of v
                                   // control points in
                                   // array
    const SPAposition ctrlpts[],   // array of control
                                   // points
    const double weights[],        // array of weights
    double point_tol,              // identical points
                                   // if within this
                                   // tolerance
    int num_knots_u,               // number of u knots
                                   // in array
    const double knots_u[],        // array of u knots
    int num_knots_v,               // number of v knots
                                   // in array
    const double knots_v[],        // array of v knots
    double knot_tol,               // identical knots if
                                   // within this
                                   // tolerance
    FACE*& face,                   // face returned
    AcisOptions* ao = NULL         // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "baseutil/logical.h"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description:

This API creates a spline surface of the specified degree in *u* and *v*. The spline surface is defined by the given sequence of control points and knots. A spline associated with this surface is used to create a FACE.

If the argument rational_u is TRUE, the surface is rational in the *u*-parameter. If rational_v is TRUE the surface is rational in the *v*-parameter.

The argument form_u specifies whether the surface is open (0), closed (1), or periodic (2) in the *u* direction. form_v specifies the same thing for the *v*-direction.

The argument pole_u indicates whether or not the surface has a singularity at the *u*-minimum or *u*-maximum parameter boundaries according to the following:

0 => no singularity at *u*-minimum or *u*-maximum boundary
1 => has a singularity at the *u*-minimum boundary
2 => has a singularity at the *u*-maximum boundary
3 => has a singularity at both boundaries

pole_v indicates the same thing for *v* parameter boundaries.

The control points are contained in the array ctrlpts. The *v* index varies first. That is, a row of *v* control points for the first *u* value is found first. Then, the row of *v* control points for the next *u* value. If the surface is rational in either parameter, it is considered a rational surface and the associated weights are in the array weights.

The tolerance point_tol determines when two control points are identical, and knot_tol performs the same function for the knot sequence.

Each of the control points should be unique, except possibly the start and end points. The knot sequence in *u* (and *v*) should form a non-negative, non-decreasing sequence with any knot value appearing at most degree_u (degree_v) times. The tolerance point_tol determines when two control points are identical, and knot_tol performs the same function for the knot sequence.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_mk_fa_spl_fit

Function: Construction Geometry, Model Topology

Action: Creates a FACE that represents a spline surface defined by a sequence of points on the surface.

Prototype:
```
outcome api_mk_fa_spl_fit (
    double fittol,              // fit tolerance
    int num_pts_u,              // number of points
                                // in u
    int num_pts_v,              // number of points
                                // in v
    const SPAposition pts[],    // array of points
    const SPAunit_vector du_s[],// start boundary
                                // conditions
    const SPAunit_vector du_e[],// end boundary
                                // conditions
    FACE*& face,                // face returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/unitvec.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: The FACE is a spline surface fitted to an *[num_pts_u]x [num_pts_v]* array of points pts.

The *v* index varies first. That is, a row of *v* control points for the first *u* value is found, then the row of *v* control points for the next *u* value.

Optional arrays of unit tangent vectors can be specified for the boundary conditions for *u* at the start (du_s) and at the end (du_e) of the surface. NULL pointers can be used for the start and end boundary conditions.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_mk_fa_spl_intp

Function: Construction Geometry, Model Topology

Action: Creates a FACE that represents a spline surface defined by a sequence of points on a surface.

Prototype:
```
outcome api_mk_fa_spl_intp (
    int num_pts_u,              // number of points
                                // in u
    int num_pts_v,              // number of points
                                // in v
    const SPAposition pts[],    // array of points
    const SPAunit_vector du_s[],// start tangent in u
    const SPAunit_vector du_e[],// end tangent in u
    const SPAunit_vector dv_s[],// start tangent in v
    const SPAunit_vector dv_e[],// end tangent in v
    FACE*& face,                // face created
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/face.hxx"
#include "baseutil/vector/position.hxx"
#include "baseutil/vector/unitvec.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates a spline surface that interpolates a set of points. The interpolation scheme is cubic in both the *u* and *v* directions.

The points are contained in the array pts. The *u* index varies first. That is, a row of *u* control points for the first *v* value is found, then the row of *u* control points for the next *v* value.

Optional arrays of unit tangent vectors can be specified for the boundary conditions for *u* at the start (du_s) and at the end (du_e) of the surface. Similarly, dv_s and dv_e specify the unit tangent conditions for *v* at the start and end. All boundary conditions must be specified or all must be NULL.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_modify_ellipse

Function: Construction Geometry, Model Geometry

Action: Modifies an argument of an ellipse.

Prototype:
```
outcome api_modify_ellipse (
    EDGE* ell,                    // elliptical edge to
                                  // modify
    const SPAposition& center,    // new center
    const SPAunit_vector& normal, // new normal
    const SPAvector& major_axis,// new major axis
    double radius_ratio,          // radius ratio
    double start_angle,           // new start angle in
                                  // radians
    double end_angle,             // new end angle in
                                  // radians
    AcisOptions* ao = NULL        // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.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: Use this API with api_get_ellipse_parameters to modify only selected parameters of the ellipse.

Errors: The entity is not an ellipse.
The entity is not top level.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_modify_line

Function: Construction Geometry, Model Geometry

Action: Modifies line end points.

Prototype:
```
outcome api_modify_line (
    EDGE* line,             // line to modify
    const SPAposition& st_pt,// new start position
    const SPAposition& end_pt,// new end position
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: Use this API with api_get_curve_ends to modify only one end point.

Errors: The entity is not a line.
The entity is not top level.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_planar_face_pr

Function: Construction Geometry, Physical Properties

Action: Locates area, center of area, second moments, and principal axes of a planar face.

Prototype:

```
outcome api_planar_face_pr (
    FACE* face,                       // planar face to
                                      // be examined
    double req_rel_accy,              // requested
                                      // relative
                                      // accuracy
    double& area,                     // area of face
                                      // returned
    SPAposition& centre,              // center of face
                                      // returned
    double& moment_a,                 // moment of
                                      // inertia about
                                      // axis a
                                      // returned
    double& moment_b,                 // moment of
                                      // inertia about
                                      // axis b
                                      // returned
    SPAunit_vector& axis_a,           // axis a
                                      // returned
    SPAunit_vector& axis_b,           // axis b
                                      // returned
    double& est_rel_accy_achieved,    // estimate of
                                      // relative
                                      // accuracy
                                      // returned
    AcisOptions* ao = NULL            // acis options
    );
```

Includes:

```
#include ”kernel/acis.hxx”
#include ”constrct/kernapi/api/cstrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”baseutil/vector/position.hxx”
#include ”baseutil/vector/unitvec.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Results are given in the local coordinate space of the body.

Errors: Pointer to face is NULL or not to a planar face.
Negative accuracy requested.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Read-only

api_q_edges_around_vertex

Function: Construction Geometry, Model Topology

Action: Gets a list of edges that share a given vertex.

Prototype:
```
outcome api_q_edges_around_vertex (
    VERTEX* vertex,          // vertex to be examined
    ENTITY_LIST* edge_list,  // list of edges returned
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: This API returns a list of edges that share a specified vertex. Uses the edges pointed to by the vertex (usually a single edge but may be multiple for a nonmanifold body) to find all coedges, next/previous coedges, and partners to reach all edges that share the vertex.

Errors: Pointer to vertex is NULL or not to a VERTEX.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Read-only

api_reverse_body

Function: Construction Geometry, Model Topology

Action: Reverses the orientations of all coedges in the body. Also reverses face orientations and the orientation of loops within the faces.

Prototype:
```
outcome api_reverse_body (
    BODY* body,              // body to be reversed
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: Refer to Action.

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

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_reverse_face

Function: Construction Geometry, Model Topology

Action: Reverses the sense of a face.

Prototype:
```
outcome api_reverse_face (
    FACE*& face,            // same face is modified
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API reverses the sense of a face; i.e., this API makes the face material void (flips the normal). Also, alters the senses of the coedges to avoid changing the shape of the face.

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

Limitations: The API does not check to see if the face belongs to a solid, nor does it compensate for side effects caused by reversal of the face. The user must ensure that the API is called only for independent faces.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_reverse_wire

Function: Construction Geometry, Model Topology

Action: Reverses the direction (sense) of a wire.

Prototype:
```
outcome api_reverse_wire (
    WIRE* wire,              // wire to reverse
    AcisOptions* ao = NULL   // acis options
    );

outcome api_reverse_wire (
    ENTITY* in_body,         // wire body to reverse
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: This API reverses the sense of a wire body by negating the sense of each coedge. If the wire has no branches or loops, and the first coedge of the wire was the start of the chain of coedges, then the first coedge of the reversed wire will be the new start of the chain of coedges. Otherwise, the wire’s first coedge will not be changed.

Errors: Entity NULL or not a wire.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_sheet_from_ff

Function: Construction Geometry, Model Topology

Action: Creates a sheet body as a copy of a face.

Prototype:
```
outcome api_sheet_from_ff (
    int num_faces,          // number of faces = 1
    FACE* faces[],          // array containing face
    BODY*& new_body,        // sheet body returned
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”constrct/kernapi/api/cstrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kerndata/top/face.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: Refer to Action.

Errors: Entity in array not a FACE.

Limitations: Currently only handles one face. The face is passed in an array to enable future expansion of functionality without changing the function signature.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_shell_external

Function: Construction Geometry, Model Topology

Action: Determines whether a shell is internal or external.

Prototype:
```
outcome api_shell_external (
    SHELL* given_shell,     // shell to be examined
    int& external,          // internal or external
                            // flag returned
    AcisOptions* ao = NULL  // acis options
    );
```

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

Description: This API determines whether a shell is internal or external. The external argument is 0 for void, 1 for external, and 2 for unknown.

Errors: Pointer to shell is NULL or not to a SHELL.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Read-only

api_solid_block

Function: Construction Geometry, Model Geometry

Action: Creates a solid block given two diagonal positions.

Prototype:
```
outcome api_solid_block (
    const SPAposition& pt1,  // first position
    const SPAposition& pt2,  // position diagonally
                             // opposite pt1
    BODY*& block,            // created block returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/body.hxx"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates a solid block aligned with the current WCS with corners at the specified points (pt1 and pt2). The edges of the block are parallel to the axes of the active WCS.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_solid_cylinder_cone

Function: Construction Geometry, Model Geometry

Action: Creates a solid cylinder or cone given two positions on the axis.

Prototype:
```
outcome api_solid_cylinder_cone (
    const SPAposition& pt1,  // position at bottom
    const SPAposition& pt2,  // position at top
    double major_radius,     // major radius at bottom
    double minor_radius,     // minor radius at bottom
    double top_radius,       // major radius at top
    const SPAposition* xpt,  // position in direction
                             // of major axis or NULL
    BODY*& cyl_or_cone,      // created cylinder or
                             // cone returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/body.hxx"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API creates a solid cylinder or cone. The cylinder or cone may be circular or elliptical.

For a circular cylinder or cone, specify the same values for major_radius and minor_radius. For an elliptical cylinder or cone, specify two different values.

For a cylinder, specify the same values for top_radius and major_radius. For a cone, specify two different values. When creating elliptical cones or cylinders, control the orientation of the major axis with xpt. If xpt is non-NULL, then xpt projects onto the plane defined by pt1 and the axis is along the major axis. If xpt is NULL, the system determines the orientation of the major axis from the active WCS.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_solid_sphere

Function: Construction Geometry, Model Geometry

Action: Creates a solid sphere given the center and radius.

Prototype:
```
outcome api_solid_sphere (
    const SPAposition& center,// center of sphere
    double radius,          // radius of sphere
    BODY*& sph,             // created sphere
                            // returned
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/body.hxx″
#include ″baseutil/vector/position.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_solid_torus

Function: Construction Geometry, Model Geometry, Model Object

Action: Creates a solid torus given the center and major and minor radii.

Prototype:
```
outcome api_solid_torus (
    const SPAposition& center,// center of torus
    double major_radius,    // major radius
    double minor_radius,    // minor radius
    BODY*& tor,             // created torus returned
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/body.hxx″
#include ″baseutil/vector/position.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API creates a solid torus given the center, major_radius, and minor_radius. The axis of the torus is parallel to the *z*-axis of the active working coordinate system.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_split_curve

Function: Construction Geometry, Model Geometry

Action: Splits a curve at a position or its intersection with another curve.

Prototype:
```
outcome api_split_curve (
    EDGE* crv,                  // curve to split
    const SPAposition* split_pt,// position to split
                                // at or NULL
    EDGE* split_crv,            // curve to split
                                // with or NULL
    ENTITY_LIST& new_crvs,      // curves created by
                                // the split returned
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/lists/lists.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: To split a curve at a position, specify the position in split_pt and NULL for split_crv.

To split a curve with a curve, specify NULL for split_pt and a pointer to the curve to split with in split_crv. This API splits the original curve at all intersections between the bounded portions of the curves. The original curve, and all new curves created as the result of splitting the curve, are added to the entity list, new_crvs.

Errors: None

Limitations: Do not specify end points of the edge; using them causes no error but returns the original curve as the only item in the entity list.

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_split_edge_at_disc

Function: Construction Geometry, Model Geometry

Action: Splits a curve at G1 or G2 discontinuities.

Prototype:
```
outcome api_split_edge_at_disc (
    EDGE* theEdge,           //edge to be split
    ENTITY_LIST& new_edges,  //resulting edges
    int cont_order           //G1 or G2
        = 1,                 //
    AcisOptions* ao          //ACIS options such as
        = NULL               //version and journal
    );
```

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

Description: The input edge is split at the G1 or G2 (disc_order = 1 or 2) by accessing the discontinuity_info stored in the curve. The result is a list of the split edges, including the original.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_terminate_constructors

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

Action: Terminates the constructor library.

Prototype:
```
outcome api_terminate_constructors ();
```

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

Description: Refer to Action.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: System routine

api_trans_edge

Function: Construction Geometry, Model Topology

Action: Creates a new edge which is a transformed copy of the specified edge.

Prototype:
```
outcome api_trans_edge (
    EDGE* edgeIn,           // edge to be copied
    SPAtransf const& t,     // transform to be
                            // applied
    EDGE*& edgeOut,         // copied edge
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″kernel/kerndata/top/edge.hxx″
#include ″baseutil/vector/transf.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: This API creates a new EDGE pointing to a new start VERTEX, a new end VERTEX, and a new CURVE, all transformed from the input edge by the specified transform.

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

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_trim_2curves

Function: Construction Geometry, Model Geometry

Action: Trims two curves to their intersection.

Prototype:
```
outcome api_trim_2curves (
    const entity_with_ray& crv1,// curve to trim plus
                                // position on part
                                // to keep
    const entity_with_ray& crv2,// curve to trim plus
                                // position on part
                                // to keep
    AcisOptions* ao = NULL      // acis options
    );
```

Includes:
```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/geomhusk/entwray.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description: This API trims the curves (crv1 and crv2) to the intersection that is closest to the average of the two rays.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_trim_chain

Function: Construction Geometry, Model Geometry

Action: Trims a list of curves so that they form a contiguous chain.

Prototype:
```
outcome api_trim_chain (
    int num_crv,                // number of curves
    const entity_with_ray* crvs,// array of curves
    logical close,              // TRUE (forms a
                                // closed chain) or
                                // FALSE
    ENTITY_LIST& trimmed_crvs,  // list of trimmed
                                // curves returned
    AcisOptions* ao = NULL      // acis options
    );
```

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

Description: This API adds all curves in the input list to the entity list trimmed_crvs. If a curve appears in the input list more than once, it is copied and the copy is added to trimmed_crvs. Each pair of curves is then trimmed using api_trim_2curves. If close is TRUE, the first and last curves are also trimmed.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_trim_curve

Function: Construction Geometry, Model Geometry

Action: Trims one end of a curve at a position or its intersection with another curve.

Prototype:
```
outcome api_trim_curve (
    const entity_with_ray& eray1,   // curve to trim
                                    // plus position
                                    // on part to
                                    // keep
    const SPAposition* trim_pt,     // position to
                                    // trim to or
                                    // NULL
    const entity_with_ray* eray2,   // curve to trim
                                    // to or NULL
                                    // returned
    AcisOptions* ao = NULL          // acis options
    );
```

Includes:
```
#include ″kernel/acis.hxx″
#include ″constrct/kernapi/api/cstrapi.hxx″
#include ″kernel/geomhusk/entwray.hxx″
#include ″kernel/kernapi/api/api.hxx″
#include ″baseutil/vector/position.hxx″
#include ″kernel/kernapi/api/acis_options.hxx″
```

Description: The curve is trimmed to either a position or at its intersection with another curve. To trim to a position, specify a position in trim_pt and NULL for eray2. The curve is trimmed to the normal projection of the position onto the curve. To trim to the intersection with another curve, specify NULL for trim_pt and an entity_with_ray for eray2. The curve is trimmed to the intersection with the curve given in eray2 that is closest to the ray given in eray2.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_trim_middle

Function: Construction Geometry, Model Geometry

Action: Trims the middle of a curve.

Prototype:

```
outcome api_trim_middle (
    const entity_with_ray& crv,     // curve to trim
                                    // plus position
                                    // on part to
                                    // trim
    const SPAposition* trim_pt1,    // position to
                                    // trim to or
                                    // NULL
    const entity_with_ray* trimr1,  // curve to trim
                                    // to or NULL
    const SPAposition* trim_pt2,    // position to
                                    // trim to or
                                    // NULL
    const entity_with_ray* trimr2,  // curve to trim
                                    // to or NULL
    EDGE*& new_edge,                // newly created
                                    // edge or NULL
                                    // returned
    AcisOptions* ao = NULL          // acis options
    );
```

Includes:

```
#include "kernel/acis.hxx"
#include "constrct/kernapi/api/cstrapi.hxx"
#include "kernel/geomhusk/entwray.hxx"
#include "kernel/kernapi/api/api.hxx"
#include "kernel/kerndata/top/edge.hxx"
#include "baseutil/vector/position.hxx"
#include "kernel/kernapi/api/acis_options.hxx"
```

Description:

Using the trim data, this API computes two trim points on the curve to be trimmed. If trim_pt1 or trim_pt2 is non-NULL, the corresponding trim point is the normal projection of the position on the curve; otherwise, the trim point is the intersection of the curves closest to the ray associated with trim_r1 or trim_r2. The action performed depends on whether the curve is closed and periodic.

For a closed periodic curve, only a single curve results from the trim. The part of the curve on which the pick was made is removed. The new_edge is NULL.

For a non-periodic curve, the part of the curve between the trim points is removed. If the trim points are both on the interior of the curve; i.e., they are not the start or end points, the trim results in two curves. The original edge is modified as the first curve, and the second curve is returned in new_edge. If one of the trim points is not in the interior of the curve, the original edge is modified, and new_edge is NULL.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_wiggle

Function: Construction Geometry, Model Object

Action: Creates a “wiggle” body for testing purposes.

Prototype:
```
outcome api_wiggle (
    double width,           // width of wiggle
    double depth,           // depth of wiggle
    double height,          // height of wiggle
    int low_v_type,         // spline shape
    int high_v_type,        // spline shape
    int low_u_type,         // spline shape
    int high_u_type,        // spline shape
    logical height_given,   // FALSE if should have
                            // no height
    BODY*& bodyOut,         // wiggle body returned
    AcisOptions* ao = NULL  // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”constrct/kernapi/api/cstrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”baseutil/logical.h”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API creates a body that contains an “interesting” spline patch for testing purposes.

Construct four splines for the edges. Each is a cubic B-spline with two spans, passing through three colinear points. The splines’ shapes are specified by the integer type argument where:

0 is a straight line
1 is a “s” shape, starting at low parameter value with a 45 degree upward (+ve z) tangent, and ending at high parameter in the same direction.
2 is a double hump, starting going upwards and ending downwards.
–1 is the same as 1, but inverted.
–2 is the same as 2, but inverted.

Errors: Width, depth, or height not greater than zero.
Invalid spline type specified.

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Changes model

api_wire_len

Function: Construction Geometry, Physical Properties

Action: Determines the length of a wire or a wire body.

Prototype:
```
outcome api_wire_len (
    BODY* body,              // given wire body
    double& length,          // length of the wire
                             // returned
    AcisOptions* ao = NULL   // acis options
    );

outcome api_wire_len (
    WIRE* wire,              // given wire
    double& length,          // length of wire
                             // returned
    AcisOptions* ao = NULL   // acis options
    );
```

Includes:
```
#include ”kernel/acis.hxx”
#include ”constrct/kernapi/api/cstrapi.hxx”
#include ”kernel/kernapi/api/api.hxx”
#include ”kernel/kerndata/top/body.hxx”
#include ”kernel/kerndata/top/wire.hxx”
#include ”kernel/kernapi/api/acis_options.hxx”
```

Description: This API determines the length of a wire body.

This API is overloaded. The version of api_wire_len which takes a WIRE instead of a BODY may also be used.

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

Limitations: Returns zero length if there is any error condition. If the given body has more than one wire, the API considers only the first wire it encounters (first version shown only).

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Read-only

api_wire_to_chain

Function: Construction Geometry, Model Topology

Action: Gets a list of the coedges of a body that is a single unbranched wire.

Prototype:
```
outcome api_wire_to_chain (
    BODY* body,              // wire body to be
                             // examined
    ENTITY_LIST& list,       // list of coedges
                             // returned
    AcisOptions* ao = NULL   // acis options
    );
```

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

Description: This API gets a list of the coedges of a body that is a single unbranched wire into a list supplied by the caller.

The sequence of the edges and coedges of the wire can be forward or backward.

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

Limitations: None

Library: constrct

Filename: cstr/constrct/kernapi/api/cstrapi.hxx

Effect: Read-only

is_PRIMITIVE_ANNOTATION

Function: Construction Geometry

Action: Determines if an ENTITY is a PRIMITIVE_ANNOTATION.

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

Includes:
```
#include ”kernel/acis.hxx”
#include ”baseutil/logical.h”
#include ”constrct/kernbody/primtive/primanno.hxx”
#include ”kernel/kerndata/data/entity.hxx”
```

Description: Refer to Action.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/kernbody/primtive/primanno.hxx

Effect: Read–only

make_face_spline

Function: Construction Geometry, Model Geometry

Action: Constructs a new FACE from a spline.

Prototype:
```
FACE* make_face_spline (
    surface const &this_surface,// The given surface
    curve const &bottom_cur,    // The low v curve
    curve const &top_cur,       // The high v curve
    curve const &left_cur,      // The low u curve
    curve const &right_cur,     // The high u curve
    SPApar_box const &pb        // Par_box to use
    );
```

Includes:

```
#include "constrct/sg_husk/face/face_cstr.err"
#include "kernel/kerndata/geom/cnstruct.hxx"
#include "kernel/kerndata/geom/intcurve.hxx"
#include "kernel/kerndata/geom/pcurve.hxx"
#include "kernel/kerndata/geom/point.hxx"
#include "kernel/kerndata/top/alltop.hxx"
#include "kernel/kerngeom/surface/spldef.hxx"
#include "kernel/kernutil/law/law.hxx"
#include "kernel/sg_husk/lawents/law_int.hxx"
#include "kernel/sg_husk/pcurve/add_pcu.hxx"
#include "kernel/spline/bs3_srf/sp3srtn.hxx"
#include "kernel/spline/sg_bs3s/sps3srtn.hxx"
```

Description: Construct a single face which is the whole of a given B–spline surface. This will handle spline surfaces closed in one or both directions, but puts prop edges at the boundaries in those cases.

In all cases, the resulting face has one loop. The number of coedges is 4 minus the number of singularities on the surface.

In the open/open case there are four edges and four vertices, in the open/closed and closed/open cases there are three edges and two vertices, and in the closed/closed case there are two edges and one vertex.

The face normal always follows the sense of the spline surface.

Errors: None

Limitations: None

Library: constrct

Filename: cstr/constrct/sg_husk/face/mk_face.hxx

Effect: Changes model