Chapter 4.
Sat Functions

Topic: Ignore

Introduction

Topic: Ignore

This chapter describes the ACIS functions that can have data appear in the SAT file. The functions are each documented in reference templates that provide a minimal description of each function and their related SAT data. The reference templates are organized alphabetically by function name.

Not all ACIS functions are documented. Only those functions that can potentially read the content of a SAT file are included.

api_get_file_info

Function: SAT Save and Restore

Action: Gets header info from the last restored file.

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

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

api_get_save_version

Function: SAT Save and Restore

Action: Gets the current save file format version.

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

Description: This API gets the output file format.

api_restore_entity_list

Function: SAT Save and Restore

Action: Restores an entity_list from disk.

Prototype:
```
outcome api_restore_entity_list (
    FILE* file_ptr,          // open file descriptor
    logical text_mode,       // TRUE if file is text,
                             // FALSE if binary
    ENTITY_LIST& entities    // returns entities made
    );
```

Description: The file pointer is an open file positioned at the point where this API begins the restore entity. When the restore is complete, the file will be correctly positioned at the end of the save entity. This allows an application to restore multiple entities intermixed with other application specific data in a single save file.

api_restore_entity_list_file

Function: SAT Save and Restore

Action: Restores an entity_list from disk.

Prototype:
```
outcome api_restore_entity_list_file (
    FileInterface* file_ptr, // open file descriptor
    ENTITY_LIST& entities    // returns entities
                             // restored
    );
```

Description: This API restores a list of entities from a file. The file_ptr points to an open file positioned at the point where this API begins the restore entity. When the restore is complete, the file will be correctly positioned at the end of the entity save. This allows an application to restore multiple entities intermixed with other application specific data in a single save file.

api_restore_entity_list_with_history

Function: SAT Save and Restore, History and Roll

Action: Restores an entity_list from disk.

Prototype:

```
outcome api_restore_entity_list_with_history (
    FILE* file_ptr,              // open file
                                 // descriptor
    logical text_mode,           // TRUE if file is
                                 // text, FALSE if
                                 // binary
    ENTITY_LIST& entities,       // returns entities
                                 // made
    HISTORY_STREAM_LIST& hslist,// returns history
                                 // streams made
    DELTA_STATE_LIST& dslist     // returns delta
                                 // states made
    );
```

Description: The file pointer is an open file positioned at the point where this API begins the restore entity. When the restore is complete, the file will be correctly positioned at the end of the save entity. This allows an application to restore multiple entities intermixed with other application specific data in a single save file.

api_restore_entity_list_with_history_file

Function: SAT Save and Restore, History and Roll

Action: Restores an entity_list from disk.

Prototype:

```
outcome api_restore_entity_list_with_history_file (
    FileInterface* file_ptr,     // open file
                                 // descriptor
    ENTITY_LIST& entities,       // returns entities
                                 // made
    HISTORY_STREAM_LIST& hslist,// returns history
                                 // streams made
    DELTA_STATE_LIST& dslist     // returns delta
                                 // states made
    );
```

Description: The file pointer is an open file positioned at the point where this API begins the restore entity. When the restore is complete, the file will be correctly positioned at the end of the save entity. This allows an application to restore multiple entities intermixed with other application specific data in a single save file.

api_save_entity_list

Function: SAT Save and Restore, Entity, Part Management

Action: Writes a list of entities to disk as text or binary.

Prototype:

```
outcome api_save_entity_list (
    FILE* file_ptr,          // open file
                             // descriptor
    logical text_mode,       // TRUE if file is text,
                             // FALSE if binary
    ENTITY_LIST const&       // returns entities
        entity_list              // to save
    );
```

Description: The file pointer argument should be an open file positioned at the point where this API begins the entity save. When the save is complete, the file will be correctly positioned at the end of the entity save; therefore, an application can save multiple bodies intermixed with other application specific data in a single save file.

api_save_entity_list_file

Function: SAT Save and Restore

Action: Writes a list of entities to disk in text or binary format.

Prototype:

```
outcome api_save_entity_list_file (
    FileInterface* file_ptr,        // open file
                                    // descriptor
    ENTITY_LIST const& entity_list  // returns
                                    // entities to be
                                    // saved
    );
```

Description: This API creates the file pointer argument an open file positioned at the point where this API begins the entity save. When the save is complete, the file will be correctly positioned at the end of the entity save; therefore, an application can save multiple bodies intermixed with other application specific data in a single save file.

api_save_entity_list_with_history

Function: SAT Save and Restore, History and Roll

Action: Writes a list of entities to disk as text or binary.

Prototype:

```
outcome api_save_entity_list_with_history (
    FILE* file_ptr,              // open file
                                 // descriptor
    logical text_mode,           // TRUE if file is
                                 // text, FALSE if
                                 // binary
    ENTITY_LIST const&           // entities to
       entity_list,              // save
    HISTORY_STREAM_LIST& hslist,// history streams to
                                 // save
    DELTA_STATE_LIST& dslist     // returns delta
                                 // states saved
    );
```

Description: The file pointer argument should be an open file positioned at the point where this API begins the entity save. When the save is complete, the file will be correctly positioned at the end of the entity save; therefore, an application can save multiple bodies intermixed with other application specific data in a single save file.

api_save_entity_list_with_history_file

Function: SAT Save and Restore, History and Roll

Action: Writes a list of entities to disk as text or binary.

Prototype:

```
outcome api_save_entity_list_with_history_file (
    FileInterface* file_ptr,        // open file
                                    // descriptor
    ENTITY_LIST const& entity_list, // entities to
                                    // save
    HISTORY_STREAM_LIST& hslist,    // history
                                    // streams to
                                    // save
    DELTA_STATE_LIST& dslist        // returns delta
                                    // states saved
    );
```

Description: The file pointer argument should describe an open file positioned at the point where this API begins the entity save. When the save is complete, the file will be correctly positioned at the end of the entity save; therefore, an application can save multiple bodies intermixed with other application specific data in a single save file.

api_save_version

Function: SAT Save and Restore

Action: Sets the save file format.

Prototype:
```
outcome api_save_version (
    int major_version,      // release number;
                            // e.g., 1
    int minor_version       // version number;
                            // e.g., 5
    );
```

Description: This API sets the output file format. For Release 1.5 and above, the system can output data in a format that a previous version can read. This is only TRUE for objects that are compatible in the previous release.

api_set_file_info

Function: SAT Save and Restore

Action: Sets header info to be written to ACIS save files.

Prototype:
```
outcome api_set_file_info (
    unsigned long,          // mask indicating fields
                            // to set
    FileInfo const& info    // info to be set
    );
```

Description: The API sets the information to be written to the header of later saved files. Does not alter the model.

BDY_GEOM_restore

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
BDY_GEOM* BDY_GEOM_restore();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restore the data from a save file. This uses a dispatch table, whereby the proper restore functions have been previously registered. At runtime, the proper restore routine is called.

```
if (form == PCURVE_BOUNDARY)
    BDY_GEOM_PCURVE::restore    Routine to perform actual work.
else if (form == PLANE_BOUNDARY)
    BDY_GEOM_PLANE::restore     Routine to perform actual work.
else if (form == CIRCLE_BOUNDARY)
    BDY_GEOM_CIRCLE::restore    Routine to perform actual work.
else if (form == DEGENERATE_BOUNDARY)
    BDY_GEOM_DEG::restore       Routine to perform actual work.
```

begin_local_savres

Function: SAT Save and Restore

Action: Starts process to allow individual items to be written to a file for debugging.

Prototype:

```
void begin_local_savres (
    FILE* fp,                // file pointer
    int major                // major release number
        = -1,
    int minor                // minor release number
        = -1
    );
```

Description: Refer to action.

bs2_curve_restore

Function: Spline Interface, Construction Geometry, SAT Save and Restore

Action: Restores a curve.

Prototype:

```
bs2_curve bs2_curve_restore ();
```

Description: Reads back a representation of a parametric curve written by bs2_curve_save and construct a duplicate of the original curve. Reading uses routines read_int, read_long, read_real, and read_string defined in kernutil/fileio/fileio.hxx.

bs_2_3_spline_restore	Information to restore from SAT

bs3_curve_restore

Function: Spline Interface, Construction Geometry, SAT Save and Restore

Action: Restores a curve from a file.

Prototype:
```
bs3_curve bs3_curve_restore ();
```

Description: Reads back a representation of a parametric curve written by bs3_curve_save and constructs a duplicate of the original curve. Reading uses routines read_int, read_long, read_real, and read_string defined in kernutil/fileio/fileio.hxx.

bs_2_3_spline_restore	Restore spline

bs3_surface_restore

Function: Spline Interface, Construction Geometry

Action: Restores a saved surface.

Prototype:
```
bs3_surface bs3_surface_restore ();
```

Description: Reads back a representation of a parametric surface as written by bs3_surface_save, and creates a duplicate of the original surface.

Reading uses routines read_int, read_long, read_real, and read_string that are defined in kernutil/fileio/fileio.hxx.

```
if (restore_version_number < SPLINE_VERSION)
    if (read_int() == –1)
        // First check that there is a surface to read.
    read_int                    stype
    read_int                    save_dim
    read_int                    u degree
    read_int                    v degree
    read_int                    save nu span
```

```
        read_int                    save nv span
        read_int                    rat u
        read_int                    rat v
        read_int                    form u
        read_int                    form v
        read_int                    pole u
        read_int                    pole v
    else
        // New style header. There are keywords instead of numbers
        // where appropriate, and redundant values are missing.
        read_id                     This class does not save any data
        if (strcmp( id_string, type_nullbs ) == 0)
            // return NULL;
        else if (strcmp( id_string, type_nubs ) == 0 )
            // rational = FALSE;
        else if (strcmp( id_string, type_nurbs ) == 0 )
            // rational = TRUE;
        else
            // sys_error( UNKNOWN_BS_SURFACE );
        read_int                    u degree
        read_int                    v degree
        if (rational)
            read_id                 id string for rational_u or
                                    rational_v
        if (restore_version_number < CONSISTENT_VERSION)
            read_id                 id string for formu
            read_id                 id string for formv
            read_id                 id string for poleu
            read_id                 id string for polev
        else
            read_enum               Read enumeration bs3_surf_form
                                    for form_map for form u
            read_enum               Read enumeration bs3_surf_form
                                    for form_map for form v
            read_enum               Read enumeration sing_map for
                                    pole u
            read_enum               Read enumeration sing_map for
                                    pole v
    // Read the knots and multiplicities, allocating space for
    // the knot values as we go, and accumulating the total of
    // knots and multiplicities.
    read_int                    Number of knots in u
    if (restore_version_number >= SPLINE_VERSION)
```

```
    read_int                        Number of knots in v
for (int i = 0; i < n_uknots; i++)
    read_real                       u knot
    read_int                        u multiplicity
if (restore_version_number < SPLINE_VERSION)
    read_int                        Number of knots in v
for (i = 0; i < n_vknots; i++)
    read_real                       v knot
    read_int                        v multiplicity
// Finally read the control point values.
for (row_start = bs–>node0;
    row_start != NULL;
    row_start = row_start–>vnext)
    for (ag_snode *this_node = row_start;
        this_node != NULL;
        this_node = this_node–>unext)
        for (i = 0; i < dimh; i++)
            read_real               node Pw weight
```

bs_2_3_spline_restore

Function: Spline Interface, SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:

```
ag_spline* bs_2_3_spline_restore (
    int dim                 // dimensionality
    );
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

```
if (restore_version_number < INTCURVE_VERSION)
     // Old style – lots of numbers.
     if (read_int() == –1)
          // First check that there is a curve to read – if so the
          // first item is 0, otherwise –1.
          // return NULL;
     read_int                         c type
     read_int                         save dimension
     read_int                         degree
     read_int                         save n span
     read_int                         rational
     read_int                         form
else
     // New style header. There are keywords instead of numbers
     // where appropriate, and redundant values are missing.
     read_id                          type: nullbs, nubs, nurbs
     if (strcmp( id_string, type_nullbs ) == 0)
          // return NULL;
     else if (strcmp( id_string, type_nubs ) == 0 )
          // rat = 0;
     else if (strcmp( id_string, type_nurbs ) == 0 )
          // rat = 1;
     else
          // sys_error( UNKNOWN_BS_CURVE )
     read_int                         degree
     if (restore_version_number < CONSISTENT_VERSION)
          read_id                     form
     else
          read_enum                   form_map
     // Read the knots and multiplicities
     read_int                         Number of knots
     for (int i = 0; i < n_knots; i++)
          read_real                   knot
          read_int                    multiplicity
     // Finally read the control point values
     // int dimh = rat ? dim + 1 : dim;
     for (this_node = bs–>node0;
          this_node != NULL;
          this_node = this_node–>next)
          for (i = 0; i < dimh; ++i)
               read_real              Control point value
```

bs_2_3_spline_save

Function: Spline Interface, SAT Save and Restore

Action: Writes a spline to the system save file.

Prototype:
```
void bs_2_3_spline_save (
    ag_spline* bs,          // input curve
    int dim                 // curve dimension
    );
```

Description: This function is also called by bs2_curve_save to save a 2D spline.

coedge_end_outdir

Function: Construction Geometry

Action: Returns the direction outwards from the surface at the end position of the coedge.

Prototype:
```
unit_vector coedge_end_outdir (
    COEDGE* coedge,         // coedge to examine
    transf const& ctrans    // transform to apply
        =*(transf*)NULL_REF, // to coedge
    FACE* face              // surface to
        = NULL,             // intersect
    transf const& ftrans    // transform to apply
        =*(transf*)NULL_REF  // to surface
    );
```

Description: This routine finds a direction outwards from the surface at a position on a coedge. This is usually the normal to the surface, but if the point is a singularity of the surface (like the apex of a cone or one apex of a degenerate torus), it just returns some direction guaranteed to point outwards from the surface (and not tangential, except for a vortex).

The start and end are obvious. The mid point is defined to be the one at middle parameter. For a parametric point, the parameter value corresponds to the parametrization of the coedge.

If the first transf is given, the result is for a coedge of a body transformed by that transf. If a face is given, the coedge is simply assumed to lie on the face, otherwise it looks for the face owning the loop of the coedge.

If the second transformation is given, this is the transform required to translate the face geometry into the same coordinate system as the untransformed coedge geometry. It should only be non–null if the face is given, and is used primarily in Boolean operations when testing a graph coedge against body faces.

coedge_mid_outdir

Function: Construction Geometry

Action: Returns the direction outwards from the surface at the mid position of the coedge.

Prototype:

```
unit_vector coedge_mid_outdir (
    COEDGE* coedge,          // coedge to examine
    transf const& ctrans     // transform to apply
        =*(transf*)NULL_REF, // to coedge
    FACE* face               // surface to
        = NULL,              // intersect
    transf const& ftrans     // transform to apply
        =*(transf*)NULL_REF  // to surface
    );
```

Description: This routine finds a direction outwards from the surface at a position on a coedge. This is usually the normal to the surface, but if the point is a singularity of the surface (like the apex of a cone or one apex of a degenerate torus), it just returns some direction guaranteed to point outwards from the surface (and not tangential, except for a vortex).

The start and end are obvious. The mid point is defined to be the one at middle parameter. For a parametric point, the parameter value corresponds to the parameterization of the coedge.

If the first transf is given, the result is for a coedge of a body transformed by that transf. If a face is given, the coedge is simply assumed to lie on the face, otherwise it looks for the face owning the loop of the coedge.

If the second transformation is given, this is the transform required to translate the face geometry into the same coordinate system as the untransformed coedge geometry. It should only be non–null if the face is given, and is used primarily in Boolean operations when testing a graph coedge against body faces.

coedge_param_outdir

Function: Construction Geometry

Action: Returns the direction outwards from the surface at the parameter position of the coedge.

Prototype:

```
unit_vector coedge_param_outdir (
    COEDGE* coedge,          // coedge to examine
    double coedge_param,     // parameter along coedge
    transf const& ctrans     // transform to apply
        =*(transf*)NULL_REF, // to coedge
    FACE* face               // surface to
        = NULL,              // intersect
    transf const& ftrans     // transform to apply
        =*(transf*)NULL_REF  // to surface
    );
```

Description: This routine finds a direction outwards from the surface at a position on a coedge. This is usually the normal to the surface, but if the point is a singularity of the surface (like the apex of a cone or one apex of a degenerate torus), it just returns some direction guaranteed to point outwards from the surface (and not tangential, except for a vortex).

The start and end are obvious. The mid point is defined to be the one at middle parameter. For a parametric point, the parameter value corresponds to the parameterization of the coedge.

If the first transf is given, the result is for a coedge of a body transformed by that transf. If a face is given, the coedge is simply assumed to lie on the face, otherwise it looks for the face owning the loop of the coedge.

If the second transformation is given, this is the transform required to translate the face geometry into the same coordinate system as the untransformed coedge geometry. It should only be non–null if the face is given, and is used primarily in Boolean operations when testing a graph coedge against body faces.

coedge_start_outdir

Function: Construction Geometry

Action: Returns the direction outwards from the surface at the starting position of the coedge.

Prototype:
```
unit_vector coedge_start_outdir (
    COEDGE* coedge,          // coedge to examine
    transf const& ctrans     // transform to apply
        =*(transf*)NULL_REF, // to coedge
    FACE* face               // surface to
        = NULL,              // intersect
    transf const& ftrans     // transform to apply
        =*(transf*)NULL_REF  // to surface
    );
```

Description: This routine finds a direction outwards from the surface at a position on a coedge. This is usually the normal to the surface, but if the point is a singularity of the surface (like the apex of a cone or one apex of a degenerate torus), it just returns some direction guaranteed to point outwards from the surface (and not tangential, except for a vortex).

The start and end are obvious. The mid point is defined to be the one at middle parameter. For a parametric point, the parameter value corresponds to the parameterization of the coedge.

If the first transf is given, the result is for a coedge of a body transformed by that transf. If a face is given, the coedge is simply assumed to lie on the face, otherwise it looks for the face owning the loop of the coedge.

If the second transformation is given, this is the transform required to translate the face geometry into the same coordinate system as the untransformed coedge geometry. It should only be non–null if the face is given, and is used primarily in Boolean operations when testing a graph coedge against body faces.

copy_body_from_body

Function: SAT Save and Restore

Action: Copies a body.

Prototype:
```
BODY* copy_body_from_body (
    BODY* body               // body to copy
    );
```

Description: Refer to action.

copy_entity_from_entity

Function: SAT Save and Restore

Action: Copies an entity structure.

Prototype:
```
ENTITY* copy_entity_from_entity (
    ENTITY* entity          // entity to copy
    );
```

Description: Refer to action.

dispatch_restore_cu

Function: SAT Save and Restore

Action: Determines which curve type to restore and calls its restore method.

Prototype:
```
curve* dispatch_restore_cu (
    char* subtype           // curve type
    );
```

Description: This function is never called directly by an application. Its purpose is to search through the list of possible curve types and then to call the appropriate restore method for the curve type passed in.

Given a curve subtype, scan the subtype list, and call the appropriate restore routine. If the type unknown, flag an error. This version is only used for old (pre–V1.8/R1.3) save files, which used the integer curve type rather than the textual name.

No data	This function does not save any data, but does route to the appropriate restore function in the curve definition table.

dispatch_restore_su

Function: SAT Save and Restore

Action: Determines which surface type to restore and calls its restore method.

Prototype:
```
surface* dispatch_restore_su (
    char* subtype           // surface type
    );
```

Description: This function is never called directly by an application. Its purpose is to search through the list of possible surface types and then to call the appropriate restore method for the surface type passed in.

No data	This function does not save any data, but does route to the appropriate restore function in the surface definition table.

dispatch_restore_subtype

Function: SAT Save and Restore

Action: Determines which subtype to restore and calls its restore method.

Prototype:
```
subtype_object* dispatch_restore_subtype (
    char const* postfix,    // postfix for name
    char const* name        // subtype name
    );
```

Description: This function is never called directly by an application. Its purpose is to search through the list of possible subtypes and then to call the appropriate restore method for the subtype passed in.

The two argument prototype calls the overloaded dispatch_restore_subtype accepting three arguments.

This routine is called when it is known that a subtype follows. It determines the beginning of the subtype definition. Then, based on the name used for the subtype identifier, it calls the appropriate restore routine for that subtype. In general, this is used for subtypes defined from int_cur and spl_sur.

Restore mechanism for subtype objects. Static declarations of objects of this class form themselves into a table containing the external (string) identifier of the particular subtype, together with a pointer to the correct restore routine. The generic restore routine reads the external identifier, and switches according to the table.

The table will probably be short, so can be simply a linear list, for ease of implementation. We keep this implementation private, so that we might some time have a more exotic version.

```
read_subtype_start              Marker indicating beginning of a
                                subtype. In the SAT file, this is a
                                "{"
read_id                         The name of the subclass identifier
if (strncmp( name, null_id, strlen( null_id ) ) == 0)
    read_subtype_end            Marker indicating ending of a
                                subtype. In the SAT file, this is a
                                "}"
else if (strcmp( name, ref_id ) == 0)
    read_int                    Index within the save for for this
                                ref_id.
    read_subtype_end            Marker indicating ending of a
                                subtype. In the SAT file, this is a
                                "}"
else
    // Not a reference, so scan the list. First try for the id as
    // read, then if unsuccessful try appending the given postfix
    // and look again.
    restore_subtype_def *this_def = search_subtype_table( name )
    // Now read the object.
    if (this_def != NULL)
        this_def–>restore       This class does not save any data
        read_subtype_end        Marker indicating ending of a
                                subtype. In the SAT file, this is a
                                "}"
    else if (unknown_types_ok() && bra_read)
        // No match found. Read it as unknown data, up to the matching
        // closing bracket.
        restore_unknown_subtype( name )
    else
        // No match found, and we are in binary mode, or the text file
        // is an old–style one without brackets. This is an error,
        // as we cannot tell the end of the unknown data.
```

edge_end_outdir

Function: Construction Geometry

Action: Returns the direction outwards from the surface at the end position of the edge.

Prototype:

```
unit_vector edge_end_outdir (
    EDGE* edge,              // edge to test
    transf const& etrans,    // edge transform
    FACE* face,              // surface to test
    transf const& ftrans     // surface transform
        =*(transf*)NULL_REF, // to edge coord system
    pcurve const& pcu        // supply for speed if
        =*(pcurve*)NULL_REF  // surface is parametric
    );
```

Description: This routine finds a direction outwards from the surface at a position on a coedge. This is usually the normal to the surface, but if the point is a singularity of the surface (like the apex of a cone or one apex of a degenerate torus), it just returns some direction guaranteed to point outwards from the surface (and not tangential, except for a vortex).

The start and end are obvious. The mid point is defined to be the one at middle parameter. For a parametric point, the parameter value corresponds to the parameterization of the coedge.

If the first transf is given, the result is for a coedge of a body transformed by that transf. If a face is given, the coedge is simply assumed to lie on the face, otherwise it looks for the face owning the loop of the coedge.

If the second transformation is given, this is the transform required to translate the face geometry into the same coordinate system as the untransformed coedge geometry. It should only be non–null if the face is given, and is used primarily in Boolean operations when testing a graph coedge against body faces.

edge_mid_outdir

Function: Construction Geometry

Action: Returns the direction outwards from the surface at the mid position of the edge.

Prototype:

```
unit_vector edge_mid_outdir (
    EDGE* edge,              // edge to test
    transf const& etrans,    // edge transform
    FACE* face,              // surface to test
    transf const& ftrans     // surface transform
        =*(transf*)NULL_REF, // to edge coord system
    pcurve const& pcu        // supply for speed if
        =*(pcurve*)NULL_REF  // surface is parametric
    );
```

Description: This routine finds a direction outwards from the surface at a position on a coedge. This is usually the normal to the surface, but if the point is a singularity of the surface (like the apex of a cone or one apex of a degenerate torus), it just returns some direction guaranteed to point outwards from the surface (and not tangential, except for a vortex).

The start and end are obvious. The mid point is defined to be the one at middle parameter. For a parametric point, the parameter value corresponds to the parameterization of the coedge.

If the first transf is given, the result is for a coedge of a body transformed by that transf. If a face is given, the coedge is simply assumed to lie on the face, otherwise it looks for the face owning the loop of the coedge.

If the second transformation is given, this is the transform required to translate the face geometry into the same coordinate system as the untransformed coedge geometry. It should only be non–null if the face is given, and is used primarily in Boolean operations when testing a graph coedge against body faces.

edge_param_outdir

Function: Construction Geometry

Action: Returns the direction outwards from the surface at the parameter position of the edge.

Prototype:

```
unit_vector edge_param_outdir (
    EDGE* edge,              // edge to test
    double edge_param,       // parameter
    transf const& etrans,    // edge transform
    FACE* face,              // surface to test
    transf const& ftrans     // surface transform
        =*(transf*)NULL_REF, // to edge coord system
    pcurve const& pcu        // supply for speed if
        =*(pcurve*)NULL_REF  // surface is parametric
    );
```

Description: This routine finds a direction outwards from the surface at a position on a coedge. This is usually the normal to the surface, but if the point is a singularity of the surface (like the apex of a cone or one apex of a degenerate torus), it just returns some direction guaranteed to point outwards from the surface (and not tangential, except for a vortex).

The start and end are obvious. The mid point is defined to be the one at middle parameter. For a parametric point, the parameter value corresponds to the parameterization of the coedge.

If the first transf is given, the result is for a coedge of a body transformed by that transf. If a face is given, the coedge is simply assumed to lie on the face, otherwise it looks for the face owning the loop of the coedge.

If the second transformation is given, this is the transform required to translate the face geometry into the same coordinate system as the untransformed coedge geometry. It should only be non–null if the face is given, and is used primarily in Boolean operations when testing a graph coedge against body faces.

edge_start_outdir

Function: Construction Geometry

Action: Returns the direction outwards from the surface at the start position of the edge.

Prototype:
```
unit_vector edge_start_outdir (
    EDGE* edge,             // edge to test
    transf const& etrans,   // edge transform
    FACE* face,             // surface to test
    transf const& ftrans    // surface transform
        =*(transf*)NULL_REF, // to edge coord system
    pcurve const& pcu       // supply for speed if
        =*(pcurve*)NULL_REF  // surface is parametric
    );
```

Description: This routine finds a direction outwards from the surface at a position on a coedge. This is usually the normal to the surface, but if the point is a singularity of the surface (like the apex of a cone or one apex of a degenerate torus), it just returns some direction guaranteed to point outwards from the surface (and not tangential, except for a vortex).

The start and end are obvious. The mid point is defined to be the one at middle parameter. For a parametric point, the parameter value corresponds to the parameterization of the coedge.

If the first transf is given, the result is for a coedge of a body transformed by that transf. If a face is given, the coedge is simply assumed to lie on the face, otherwise it looks for the face owning the loop of the coedge.

If the second transformation is given, this is the transform required to translate the face geometry into the same coordinate system as the untransformed coedge geometry. It should only be non–null if the face is given, and is used primarily in Boolean operations when testing a graph coedge against body faces.

end_local_savres

Function: SAT Save and Restore

Action: Terminates local saving.

Prototype:
```
void end_local_savres ();
```

Description: Refer to action.

ENTITY_restore_data

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
ENTITY* ENTITY_restore_data ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

ENTITY_restore_data works just like any derived class' restore function, but it is called explicitly by restore_entity_from_file if no id string is recognized. ENTITY::restore_common is also just like one for a derived class, except that, of course, it cannot call the function for its parent class.

ENTITY::restore_common	Calls the common restore method for the entities.

find_entity_code

Function: SAT Save and Restore

Action: Gets the integer identifier of the entity described by the given external identifier.

Prototype:
```
int find_entity_code (
    const char* entity_str  // external ID
    );
```

Description: The identifier is also truncated to just the portion that cannot be matched, separating the "–" unrecognized portion form the recognized part. If nothing is recognized, this method returns 0 and the input string does not change.

find_restore_def

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
restore_def const* find_restore_def (
    char* entity_str        // string to search on
    );
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Searches the restore definition structure to find the entry corresponding to a given external ENTITY identifier string. The string consists of identifiers separated by "–", with base identifier last and leaf identifier first. It should contain no white space. The function finds the restore definition object which matches the longest right–to–left sequence of identifiers, and modifies the input string by replacing the "–" immediately before the matched string with a terminator, or making the whole string empty if it is all matched.

No data	This function does not save any data, but does route to the appropriate restore function in the definition table.

get_file_info

Function: SAT Save and Restore

Action: Retrieves information about file.

Prototype:
```
void get_file_info (
    FileInfo& info          // file information
    );
```

Description: Refer to action.

get_save_file_version

Function: SAT Save and Restore

Action: Gets the save / restore file version.

Prototype:
```
void get_save_file_version (
    int& major,             // major release number
    int& minor              // minor release number
    );
```

Description: Refer to action.

get_savres_file

Function: SAT Save and Restore

Action: Gets the file interface object corresponding to the current SAT file.

Prototype:
```
FileInterface* get_savres_file ();
```

Description: Refer to Action.

get_savres_file_interface

Function: SAT Save and Restore

Action: Gets the save / restore file interface in use.

Prototype:
```
FileInterface* get_savres_file_interface (
    FILE* file_ptr,          // file pointer
    logical mode_text        // SAT or SAB
    );
```

Description: Refer to action.

LW_REFINEMENT_restore_data

Function: Faceting, SAT Save and Restore

Action: Saves data for saving and restoring refinements.

Prototype:
```
ENTITY* LW_REFINEMENT_restore_data ();
```

Description: This is used for saving and restoring refinements. This should not be called directly.

ENTITY::restore_common	REFINEMENT is derived from ENTITY. Create an instance of this class but then use the inherited ENTITY restore_common method.
read_int	Minimum level (ignored)
read_int	maximum grid lines
read_real	flatness tolerance
read_real	silhouette tolerance
read_real	surface tolerance
read_real	normal tolerance
read_real	pixel area tolerance
read_real	grid aspect ratio
read_int	mode

read_array

Function: SAT Save and Restore

Action: Reads an of array indices.

Prototype:
```
ENTITY* read_array (
    ENTITY* array[],          // array of entities
    int i                     // number of entities
    );
```

Description: This routine is used as part of restore from a SAT or SAB file. It returns an array of indices or NULL for negative index.

```
if (i < 0)
    return NULL
else
    return array[i]                    Array of indices.
```

read_char

Function: SAT Save and Restore

Action: Reads a character written with C printf format “%c”.

Prototype:
```
int read_char ();
```

Description: This routine is used as part of restore from a SAT or SAB file. ActiveFile is a FileInterface object and does most of the actual work.

```
return ActiveFile ? ActiveFile–>read_char() : EOF;
                                    Call the appropriate SatFile or
                                    SabFile method
```


read_data

Function: SAT Save and Restore

Action: Reads a TaggedData item from an unkown ENTITY type.

Prototype:
```
TaggedData* read_data ();
```

Description: This routine is used as part of restore from a SAT or SAB file. ActiveFile is a FileInterface object and does most of the actual work. Reads a TaggedData item from an unkown ENTITY type. This procedure returns a new object which is allocated on the heap. It is the callers responsibility to free it when it is done with it. Normally, the object will be appended to a TaggedDataList, and the list will assume responsibility for deleting it.

return ActiveFile ? ActiveFile–>read_data() : NULL;
Call the appropriate SatFile or SabFile method

read_enum

Function: SAT Save and Restore

Action: Reads an enumeration table.

Prototype:
```
int read_enum (
    enum_table const& tbl    // enumeration table
    );
```

Description: Read an enumeration table. The <identifier> specifies which enumeration is active and its valid values. The <identifier> is not written to the file. A valid value only is written to the file. This is a character string or a long value from the enumeration <identifier> written with C printf format "%s". For compatibility with older files, accept the integer value, even for interfaces which write the corresponding string. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_enum(tb1) : 0;

read_float

Function: SAT Save and Restore

Action: Reads a float written with C printf format "%g ".

Prototype:
```
float read_float ();
```

Description: This routine is used as part of restore from a SAT or SAB file. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_float() : 0;
Call the appropriate SatFile or SabFile method

read_header

Function: SAT Save and Restore

Action: Reads a header.

Prototype:
```
logical read_header (
    int& i1,                // release level
    int& i2,                // number of data records
    int& i3,                // number of entities
    int& i4                 // history
    );
```

Description: Reads a header. The first record of the ACIS save file is a header, such as: 200 0 1 0

First Integer: An encoded version number. In the example, this is “200”. This value is 100 times the major version plus the minor version (e.g., 107 for ACIS version 1.7). For point releases, the final value is truncated. Part save data for the .sat files is not affected by a point release (e.g., 105 for ACIS version 1.5.2).

Second Integer: The total number of saved data records, or zero. If zero, then there needs to be an end mark.

Third Integer: A count of the number of entities in the original entity list saved to the part file.

Fourth Integer: The least significant bit of this number is used to indicate whether or not history has been saved in this save file.

ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_header(i1, i2, i3, i4) : FALSE;
Call the appropriate SatFile or SabFile method

read_id

Function: SAT Save and Restore

Action: Reads an identifier.

Prototype:
```
int read_id (
    char* buf,              // id string
    int buflen              // length of buffer
        = 0
    );
```

Description: The save identifier written with C printf format “%s ”. Read an entity identifier. In text mode, this is just a sequence of non–blank characters. In binary mode, it is a sequence of counted strings, of which all but the last have negative counts. These strings are assembled into the buffer, separated by ’–’. The result is placed in a caller–supplied buffer – overflow causes an error, unless the length is given zero or negative, in which case no overflow is detected. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_id(buf, buflen) : 0;
Call the appropriate SatFile or SabFile method

read_int

Function: SAT Save and Restore

Action: Reads an integer by reading a long and converting.

Prototype: `int read_int ();`

Description: This routine is used as part of restore from a SAT or SAB file. Reads an integer by reading a long and converting. Some compilers will give a warning for this shortening, but it may be ignored. Implementations for machines with ints and longs different lengths may well want a different version. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? (int)(ActiveFile–>read_long()) : 0;
Call the appropriate SatFile or SabFile method

read_interval

Function: SAT Save and Restore

Action: Reads an interval as two doubles.

Prototype: `interval read_interval ();`

Description: This routine is used as part of restore from a SAT or SAB file. Reads an interval as two doubles (old–style), or as two instances of ”I” for infinite, or as ”F <value>” for finite bound.

```
if (restore_version_number < INFINT_VERSION)
    read_real                       starting
    read_real                       ending
else
    read_logical                    finite: either “I” or “F”
    if (finite)
        read_real                   ending
```

read_logical

Function: SAT Save and Restore

Action: Reads a logical.

Prototype:

```
logical read_logical (
    char const* false_str    // string for FALSE
        = ”F”,
    char const* true_str     // string for TRUE
        = ”T”
    );
```

Description: (false_string, true_string, {or any_valid_string}): Appropriate string written with C printf format “%s ”. Reads a logical value. Up to LOGICAL_VERSION, this was an integer 0 or 1. Later than that in text files it has been keywords defaulting to ”T” or ”F”. For generality, accept an integer value or any blank–terminated string starting with the first character of either of the given strings. ActiveFile is a FileInterface object and does most of the actual work.

```
return ActiveFile ? ActiveFile–>read_logical(false_str, true_str) : FALSE;
                                    Call the appropriate SatFile or
                                    SabFile method
```

4

read_long

Function: SAT Save and Restore

Action: Reads a long written with C printf format “%ld”.

Prototype:

```
long read_long ();
```

Description: This routine is used as part of restore from a SAT or SAB file. Reads a long integer. In text mode, this ignores initial white space, and leaves the input stream positioned at the character (which should be white space) which terminates the decimal integer representation. In binary, this simply reads the correct number of bytes for the internal representation, and then possibly reorders them. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_long() : 0;
Call the appropriate SatFile or SabFile method

read_matrix

Function: SAT Save and Restore, Mathematics

Action: Reads a matrix as three row vectors.

Prototype: `matrix read_matrix ();`

Description: This routine is used as part of restore from a SAT or SAB file.

read_vector	vector v1
read_vector	vector v2
read_vector	vector v3

read_pointer

Function: SAT Save and Restore

Action: Reads a pointer.

Prototype: `void* read_pointer ();`

Description: Reads a pointer. Pointer reference to a save file record index. Written as "$" followed by index number written as a long. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_pointer() : NULL;
Call the appropriate SatFile or SabFile method

read_position

Function: SAT Save and Restore

Action: Reads a position as three doubles.

Prototype: `position read_position ();`

Description: This routine is used as part of restore from a SAT or SAB file. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_position() : position(0,0,0);
Call the appropriate SatFile or SabFile method

read_ptr

Function: SAT Save and Restore

Action: Reads a pointer for the save file.

Prototype: `ENTITY* read_ptr ();`

Description: This routine is used as part of restore from a SAT or SAB file.

return (ENTITY *)read_pointer(); Call the other read pointer function.

read_real

Function: SAT Save and Restore

Action: Reads a double.

Prototype: `double read_real ();`

Description: This routine is used as part of restore from a SAT or SAB file. Read a double. In text mode, this ignores initial white space, and leaves the input stream positioned at the character (which should be white space) which terminates the decimal representation, which may be fixed–point or exponent notation. In binary, this simply reads the correct number of bytes for the internal representation, and then possibly reorders them. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_double() : 0;
Call the appropriate SatFile or SabFile method

read_sequence

Function: SAT Save and Restore

Action: Reads an explicit record sequence number.

Prototype: `int read_sequence ();`

Description: This routine is used as part of restore from a SAT or SAB file. Reads an explicit record sequence number, returning it, or negative if none. Sequence numbers in text mode consist of a minus sign with no preceding white space, followed by a positive or zero integer. They do not appear in binary files. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_sequence() : –1;
Call the appropriate SatFile or SabFile method

read_string

Function: SAT Save and Restore

Action: Reads a string into a supplied buffer of a given size, maxlen.

Prototype:
```
char* read_string (
    int& len                // length of buffer
    );
```

Description: This routine is used as part of restore from a SAT or SAB file. Reads a string. This consists of an integer length, followed by that number of literal characters. In text mode, the length and characters are separated by exactly one space. In int read_string, we assume that the buffer supplied is of sufficient length for the characters plus the usual terminating null. The function returns the actual number of characters read. The char* read_string is a more convenient form of read_string. The string is written the same as it was for the old version, with a count followed by the actual string. Unlike the old version however, this version allocates a string of the correct length and returns a pointer to it, so you do not have to worry about reading the count, and then backspacing the file to re–read the string if you want to make sure that you have a buffer which is big enough. If the length of the string was zero characters, then this will return NULL rather than "". ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_string(buf) : 0;
Call the appropriate SatFile or SabFile method

return ActiveFile ? ActiveFile–>read_string(len) : NULL;
Call the appropriate SatFile or SabFile method

read_subtype_end

Function: SAT Save and Restore

Action: Reads subtype end, braces around the subtypes, written as "} ".

Prototype:
```
logical read_subtype_end ();
```

Description: This routine is used as part of restore from a SAT or SAB file. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_subtype_end() : FALSE;
Call the appropriate SatFile or SabFile method

read_subtype_start

Function: SAT Save and Restore

Action: Reads subtype start,braces around the subtypes, written as "{ ".

Prototype:
```
logical read_subtype_start ();
```

Description: This routine is used as part of restore from a SAT or SAB file. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_subtype_start() : FALSE;
Call the appropriate SatFile or SabFile method

read_transf

Function: SAT Save and Restore, Mathematics, Transforms

Action: Internal to ACIS and not intended for direct usage. Reads a transformation.

Prototype:
```
transf read_transf ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Read a transformation as matrix, translation vector, double scaling factor and three integer flags.

read_matrix	Affine matrix
read_vector	Translation vector
read_real	Scaling
read_logical	Either “no_rotate” or “rotate”
read_logical	Either “no_reflect” or “reflect”
read_logical	Either “no_shear” or “shear”

read_unit_vector

Function: SAT Save and Restore

Action: Reads a unit vector as a vector and then normalizes it.

Prototype:
```
unit_vector read_unit_vector ();
```

Description: This routine is used as part of restore from a SAT or SAB file. Reads a unit vector as a vector and then normalizes it.

read_vector	Vector to read in.

read_vector

Function: SAT Save and Restore

Action: Reads a vector as three doubles.

Prototype:
```
vector read_vector ();
```

Description: This routine is used as part of restore from a SAT or SAB file. ActiveFile is a FileInterface object and does most of the actual work.

return ActiveFile ? ActiveFile–>read_vector() : vector(0,0,0);
Call the appropriate SatFile or SabFile method

restore_BDY_GEOM

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
BDY_GEOM* restore_BDY_GEOM();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

```
if (restore_version_number < CONSISTENT_VERSION) {
    read_int                        form
    if (form == PCURVE_BOUNDARY)
        BDY_GEOM_PCURVE::restore
                                    Routine to perform actual work.
    else if (form == PLANE_BOUNDARY)
        BDY_GEOM_PLANE::restore
                                    Routine to perform actual work.
    else if (form == CIRCLE_BOUNDARY)
        BDY_GEOM_CIRCLE::restore
                                    Routine to perform actual work.
    else if (form == DEGENERATE_BOUNDARY)
        BDY_GEOM_DEG::restore       Routine to perform actual work.
else
    BDY_GEOM_restore                Use dispatch table.
```

restore_BDY_GEOM_CIRCLE

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
BDY_GEOM*  restore_BDY_GEOM_CIRCLE();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

```
BDY_GEOM_CIRCLE::restore        Routine to perform actual work.
```

restore_BDY_GEOM_DEG

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
BDY_GEOM* restore_BDY_GEOM_DEG();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

BDY_GEOM_DEG::restore	Routine to perform actual work.

restore_BDY_GEOM_PCURVE

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
BDY_GEOM* restore_BDY_GEOM_PCURVE();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

BDY_GEOM_PCURVE::restore	Routine to perform actual work.

restore_BDY_GEOM_PLANE

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
BDY_GEOM* restore_BDY_GEOM_PLANE();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

BDY_GEOM_PLANE::restore	Routine to perform actual work.

restore_blend_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_blend_int_cur ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

blend_int_cur::restore_data	Routine to perform actual work.

restore_blend_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_blend_spl_sur ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

None	Nothing is saved or restored.

restore_body_from_file

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
BODY* restore_body_from_file (
    FILE* file_ptr,          // pointer to file
    logical mode_text        // text or binary
    );
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Reads the body from the file, in text or binary.

restore_entity_from_file	Routine to perform actual work.

restore_compcurv

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
curve* restore_compcurv ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the compcurv. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough.

compcurv::restore_data — Restores the low-level geometry for the compcurv.

restore_com_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
com_cur* restore_com_cur ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the com_cur. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough.

com_cur::restore_data — Restore the underlying com_cur.

restore_cone

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
surface* restore_cone ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores a cone.

if (restore_version_number < SURFACE_VERSION)

read_int — The curve type of the base ellipse used to be saved, though it is redundant

cone::restore_data — Save the rest of the cone data.

restore_cross_section

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `var_cross_section* restore_cross_section ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

var_cross_section::restore_data — Routine to perform actual work

restore_crv_crv_v_bl_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_crv_crv_v_bl_spl_sur ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores a crv_crv_v_bl_spl_sur.

crv_crv_v_bl_spl_sur::restore_data — Save the rest of the data.

restore_crv_srf_v_bl_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_crv_srf_v_bl_spl_sur ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores a crv_srf_v_bl_spl_sur.

crv_srf_v_bl_spl_sur::restore_data — Save the rest of the data.

restore_curve

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
curve* restore_curve ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the curve. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough. This reads the curve type and then switches in the run-time table to the correct restore routine.

if (restore_version_number < CURVE_VERSION)

read_int	integer for the type of curve.
dispatch_restore_cu	Supply the number for the type of curve

else

read_id	Reads in the string associated with the curve identification.
dispatch_restore_cu	Supply the curve identification for the type of curve

restore_degenerate_curve

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
curve* restore_degenerate_curve ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restore the data for a degenerate_curve from a save file.

degenerate_curve::restore_data	Call the method to restore the bulk of the curve data.

restore_ellipse

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
curve* restore_ellipse ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the ellipse. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough.

ellipse::restore_data	Calls the method for doing the actual work.

restore_entity_from_file

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
ENTITY* restore_entity_from_file (
    FILE* file_ptr,          // file pointer
    logical mode_text        // text or binary
    );
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the entity structure from the file, in text or binary.

restore_entity_list_from_file	Calls the routine for doing the actual work.

restore_entity_list_from_file

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
logical restore_entity_list_from_file (
    FILE* file_ptr,          // input file
    logical mode_text,       // type of file, SAT or
                             // SAB
    ENTITY_LIST& entities    // list to restore
    );
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores an entity list from a SAT file.

```
// Prototype with three arguments: FILE, logical, ENTITY_LIST
restore_entity_list_from_file          Reads in entity list from file

// Prototype with two arguments: FILE, restore_data
restore_some_entities Reads in entity list from file

// Prototype with two arguments: FILE, ENTITY_LIST
restore_entity_list_from_file          Reads in entity list from file.
// "r" is set upon successful completion of restore_entity_list_from_file
if( r && rd.history_flag )
    read_id                            Reads in entity list from file
    if(strcmp(id_array, ACIS_EOF) == 0)
        break
    if(strcmp(id_array, ACIS_HISTORY_EOS) == 0)
        ENTITY::restore_common         Perform other read operations
        ENTITY::restore_end            Finish restore process
```

restore_entity_list_from_file_with_history

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:

```
logical restore_entity_list_from_file_with_history (
    FILE* file_ptr,          // file pointer
    logical mode_text,       // SAT or SAB flag
    ENTITY_LIST& entities,   // entity list
    HISTORY_STREAM_LIST& histories,// histories
    DELTA_STATE_LIST& dslist // delta state
    );
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores an entity list from a SAT file.

```
// Prototype with five arguments:
restore_entity_list_from_file_with_history
                                        Calls prototype with four
                                        arguments
// If the restore failed and we are in binary mode, it could be
// because the file is an old style binary file. Try again with
// the old binary FileInterface.
if( !ok && !mode_text )
    restore_entity_list_from_file_with_history
                                        Calls prototype with four
                                        arguments

// Prototype with four arguments:
restore_entity_list_from_file           Reads in entity list from file.
// "r" is set upon successful completion of restore_entity_list_from_file
if( r && rd.history_flag )
    read_id                             Reads in entity list from file
    while (1)
        if(strcmp(HISTORY_STREAM_NAME, id_array) == 0)
            HISTORY_STREAM::restore
        else if(strcmp(id_array, DELTA_STATE_NAME) == 0)
            DELTA_STATE::restore
        if(strcmp(id_array, ACIS_HISTORY_EOS) == 0)
            break
    restore_some_entities               Finish restore process
```

restore_exact_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_exact_int_cur();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the exact_int_cur. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough.

exact_int_cur::restore_data	Call the restore method which does the actual work.

restore_exact_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_exact_spl_sur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the exact_spl_sur. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough.

exact_spl_sur::restore_data	Call to the restore routine that does most of work.

restore_exp_par_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_exp_par_cur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the exp_par_cur. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough.

exp_par_cur::restore_data	Call method to perform the actual work.

restore_imp_par_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_imp_par_cur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the imp_par_cur. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough.

imp_par_cur::restore_data	Routine to perform actual work.

restore_intcurve

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
curve* restore_intcurve ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the intcurve. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough.

intcurve::restore_data Restore function to do actual work

restore_int_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_int_int_cur ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

The restore function for int_int_cur is special, as it has to handle old-style SAT files, where that form was used for exact and surface int_curs as well. As a result, it has to get at the surface and pcurve pointers.

int_int_cur::restore_data Routine to perform actual work.

restore_law

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
law* restore_law ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

This extracts a law from the save file from the current read location. This is only used in an application that is reading a save file (.sat or .sab).

read_string	Associated law string within a set of double quotation marks. Law strings can be any valid combination of law symbols.
read_int	The number of law data items (dsize) attached to law definition.
for(i=0;i<dsize;i++) law_data* restore_law_data	Restore the individual law data items.

restore_law_data

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `law_data* restore_law_data ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

This extracts a law from the save file from the current read location. This is only used in an application that is reading a save file (.sat or .sab).

```
read_string                          String represents the type of law
                                     data that appears after law
                                     definition.
if(strcmp(type,"TRANS")==0)
    read_transf                      Read in the associated
                                     TRANSFORM.
else  if(strcmp(type,"WIRE")==0)
    read_int                         The number of WIRE instances to
                                     restore. (Represents size of an
                                     array).
    for(int  i=0;i<size;i++)
        curve* restore_curve         Restore the underlying curve.
        read_real                    Starting parameter for curve.
        read_real                    Scale factor.
        read_interval Range for the curve.
else  if(strcmp(type,"EDGE")==0)
    curve* restore_curve             Restore the underlying curve.
    read_real                        Starting parameter for curve.
    read_real                        Ending parameter for curve.
else  if(strcmp(type,"SURF")==0)
    surface* restore_surface         Restore the underlying surface.
    read_interval      u domain for surface.
    read_interval      v domain for surface.
else  if(strcmp(type,"PCURVE")==0)
    pcurve* restore_pcurve           Restore the underlying pcurve.
    read_real                        Starting parameter for pcurve.
    read_real                        Ending parameter for pcurve.
else                                 System error: unknown law data
                                     type.
```

restore_law_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_law_int_cur();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Constructs an law_int_cur, then calls the appropriate method to do the actual work.

law_int_cur::restore_data	Restore method to do actual work.

restore_law_par_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_law_par_cur ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Constructs an law_par_cur, then calls the appropriate method to do the actual work.

law_par_cur::restore_data	Restore method to do actual work.

restore_law_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_law_spl_sur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Constructs a law_spl_sur, then calls the appropriate method to do the actual work.

law_spl_sur::restore_data	Call the method to do actual work.

restore_meshsurf

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
surface* restore_meshsurf ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the meshsurf. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough.

meshsurf::restore_data	Call method to perform actual work.

restore_net_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_net_spl_sur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restore routine for retrieving the data for a net_spl_sur from a save file.

net_spl_sur::restore_data	The routine to perform the actual work.

restore_offset_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_offset_int_cur ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restore routine to retrieve the data for a offset_int_cur from a save file.

offset_int_cur::restore_data	Call the routine to perform the actual work.

restore_off_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_off_int_cur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the data for a off_int_cur from a save file.

off_int_cur::restore_data — Routine to perform actual work.

restore_off_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_off_spl_sur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restore routine to retrieve the data for a off_spl_sur from a save file.

off_spl_sur::restore_data — Routine to perform actual work.

restore_off_surf_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_off_surf_int_cur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restore routine to retrieve the data for a off_surf_int_cur from a save file.

off_surf_int_cur::restore_data Routine to perform actual work.

restore_old_bl_edge_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_old_bl_edge_int_cur ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

restore_old_var_rad_spl

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_old_var_rad_spl ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly. This function does the registration for the string "varblndsur" during the restore.

Restores the srf_srf_v_vl_spl_sur. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough.

restore_one_entity

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:

```
LOCAL_PROC logical restore_one_entity (
    char* id_array,          // input file
    ENTITY*& new_ent         // entity to restore
    );
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores an entity list from a SAT file.

```
find_restore_def                    for the id_array passed in
if (found_restore_def == NULL)
    ENTITY::restore_common          Restore vanilla ENTITY.
else
    // Call restore routine to read as much data as possible.
    get_restore_routine
ENTITY::restore_end Finish restore process
```

restore_ortho_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_ortho_spl_sur ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restore routine for retrieving the data for a ortho_spl_sur from a save file.

ortho_spl_sur::restore_data	Routine to perform the actual work.

restore_para_silh_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_para_silh_int_cur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restores routine to retrieve the data for a para_silh_int_cur from a save file.

para_silh_int_cur::restore_data	Routine to perform the actual work.

restore_par_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_par_int_cur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restore routine to retrieve the data for a par_int_cur from a save file.

par_int_cur::restore_data	Routine to perform the actual work.

restore_pcurve

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
pcurve* restore_pcurve ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restore routine to retrieve the data for a pcurve from a save file.

read_id	Subtype reference identifier
if (strcmp(id, pcurve_id) == 0)	
pcurve::restore_data	Routine to perform actual work.

restore_persp_silh_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_persp_silh_int_cur();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restore routine for retrieving the data for a persp_silh_int_cur from a save file.

persp_silh_int_cur::restore_data	Routine to perform actual work.

restore_pipe_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_pipe_spl_sur();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restore routine for retrieving the data for a pipe_spl_sur from a save file.

pipe_spl_sur::restore_data	Routine to perform the actual work.

restore_plane

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
surface* restore_plane();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the cone. The restore function does the actual work. It calls the base class, then reads the selector, if the save file is new enough.

plane::restore_data — Routine to perform actual work.

restore_pre_30

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_pre_30 ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly. Used to save to pre-ACIS 3.0 versions.

surface * restore_surface	specific surface data
curve * restore_curve	specific interpolated curve data
read_unit_vector	direction of taper
read_real	sine of angle
read_real	cosine of angle
read_inteval	u range
read_inteval	v range
read_int	u closure form; either “open”, “closed”, “periodic”, or “unknown”.

restore_proj_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_proj_int_cur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the data for a proj_int_cur from a save file.

proj_int_cur::restore_data	Routine to perform actual work.

restore_radius

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
var_radius* restore_radius ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

```
if (restore_version_number < CONSISTENT_VERSION)
    read_int                          form or type of radius.
    if ( form == two_ends_form )
        var_rad_two_ends::restore_data
    else if ( form == functional_form )
        var_rad_functional::restore_data
    else if ( form == fixed_width_form )
        var_rad_fixed_width::restore_data
    else if ( form == rot_ellipse_form )
        var_rad_rot_ellipse::restore_data
else
    var_radius_restore
```

restore_rb_blend_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_rb_blend_spl_sur ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

blend_spl_sur::restore_data Routine to perform actual work.

restore_rot_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_rot_spl_sur();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

rot_spl_sur::restore_data Routine to perform actual work.

restore_ruled_tpr_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_ruled_tpr_spl_sur ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

ruled_tpr_spl_sur::restore_data Routine to perform actual work.

restore_sfcv_free_bl_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_sfcv_free_bl_spl_sur ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

sfcv_free_bl_spl_sur::restore_data	Routine to perform actual work but will use parent's version.

restore_shadow_tpr_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_shadow_tpr_spl_sur ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

shadow_tpr_spl_sur::restore_data	Routine to perform actual work.

restore_skin_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_skin_spl_sur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

skin_spl_sur::restore_data — Routine to perform actual work

restore_some_entities

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
logical restore_some_entities (
    restore_data& rd        // pointer to data
    );
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restores the entity list.

```
for (;;)
    read_sequence                   Read in the sequence number
    read_id                         id of entity to restore
    // Check to see if this is the end of the data
    if(rd.num_ents_to_restore == 0)
        if(strcmp(id_array, ACIS_EOF) == 0)
            break                   Nothing is saved or restored.
    // Check for Begin of History section
    if(strcmp(id_array, ACIS_HISTORY_BEGIN) == 0)
        break                       Nothing is saved or restored.
    restore_one_entity              Restore an individual entity
```

restore_sphere

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
surface* restore_sphere();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restore routine to retrieve a sphere from a save file. This is never called directly.

sphere::restore_data	Routine to perform actual work.

restore_spline

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
surface* restore_spline ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Primary restore routine to retrieve a spline from a save file. This is never called directly.

spline::restore_data	Routine to perform actual work

restore_spring_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_spring_int_cur ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

spring_int_cur::restore_data Routine to perform actual work

restore_srf_srf_v_bl_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_srf_srf_v_bl_spl_sur ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly. Even though there is no new data, restore_srf_srf_v_bl_spl_sur is implemented, because it does a ”new”. The parent’s version of everything else can be used.

srf_srf_v_bl_spl_sur::restore_data Routine to perform actual work

restore_straight

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `curve* restore_straight ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

straight::restore_data Routine to perform actual work.

restore_stripc

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `surface* restore_stripc();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

stripc::restore_data Routine to perform actual work.

restore_subset_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_subset_int_cur();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

subset_int_cur::restore_data	Routine to perform actual work.

restore_sub_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_sub_spl_sur();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

sub_spl_sur::restore_data	Routine to perform actual work

restore_sum_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_sum_spl_sur();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

sum_spl_sur::restore_data	Routine to perform actual work

restore_surface

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
surface* restore_surface ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

```
if (restore_version_number < SURFACE_VERSION)
    // Old style: first item is the integer surface type
    read_int                        Read the type of surface
    dispatch_restore_su( type )     Restore that type of surface
else
    read_id                         Read the type of surface
    dispatch_restore_su( type )     Restore that type of surface
```

restore_surf_int_cur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_surf_int_cur();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

surf_int_cur::restore_data — Routine to perform actual work.

restore_sweep_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_sweep_spl_sur();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

sweep_spl_sur::restore_data — Routine to perform actual work.

restore_swept_tpr_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_swept_tpr_spl_sur ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

swept_tpr_spl_sur::restore_data Routine to perform actual work.

restore_taper_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_taper_spl_sur ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly. It is used for old style taper_spl_sur.

restore_pre_30 Routin to check on version

restore_torus

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `surface* restore_torus();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

torus::restore_data	Routine to perform actual work.

restore_tri3_msh_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
msh_sur* restore_tri3_msh_sur ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

tri3_msh_sur::restore_data	Routine to perform actual work.

restore_tube_spl_sur

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_tube_spl_sur();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

tube_spl_sur::restore_data	Routine to perform actual work.

restore_undefc

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
curve* restore_undefc ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

undefc::restore_data	Routine to perform actual work.

restore_unknown_entity_text

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
unknown_entity_text*
    restore_unknown_entity_text (
    char const* name        // name to use
    );
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Used as part of the save and restore operation. This is never called by an application directly.

unknown_entity_text::data_list::restore — unknown_entity_text uses the data_list method, which calls its restore method– the routine to perform the actual read.

restore_var_rad_fixed_width

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
static var_radius *restore_var_rad_fixed_width();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restore the data for a var_rad_functional from a save file.

var_rad_fixed_width::restore_data — Routine to perform actual work.

restore_var_rad_functional

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
static var_radius *restore_var_rad_functional ();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restore the data for a var_rad_functional from a save file.

var_rad_functional::restore_data Routine to perform actual work.

restore_var_rad_rot_ellipse

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `static var_radius *restore_var_rad_rot_ellipse();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restore the data for a var_rad_rot_ellipse from a save file.

var_rad_rot_ellipse::restore_data Routine to perform actual work.

restore_var_rad_two_ends

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `static var_radius *restore_var_rad_two_ends();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Restore the data for a var_rad_two_ends from a save file.

var_rad_two_ends::restore_data Routine to perform actual work.

restore_VBL_OFFSURF

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype: `subtype_object* restore_VBL_OFFSURF ();`

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

Function to search the restore definition structure to find the entry corresponding to a given external ENTITY identifier string. The string consists of identifiers separated by '–', with base identifier last and leaf identifier first. It should contain no white space. The function finds the restore definition object which matches the longest right–to–left sequence of identifiers, and modifies the input string by replacing the '–' immediately before the matched string with a terminator, or making the whole string empty if it is all matched.

Restore the data from a save file.

VBL_OFFSURF::restore_data Routine to perform actual work.

restore_VBL_SURF

Function: SAT Save and Restore

Action: Internal to ACIS and not intended for direct usage.

Prototype:
```
subtype_object* restore_VBL_SURF();
```

Description: Although this internal function is intended strictly for ACIS usage, a minimal amount of information about this function is provided for the sole purpose of being able to understand and trace restoration from a SAT file. This function should never be called directly, because it makes assumptions about the availability of a SAT file, the location of the input pointer into the SAT file, and the validity of SAT data it expects to read in. It also may start a lengthy process of nested function or class method calls, which have many of the same assumptions.

VBL_SURF::restore_data	Routine to perform actual work.

save_body_on_file

Function: SAT Save and Restore

Action: Saves the body in a file, in text or binary.

Prototype:
```
logical save_body_on_file (
    FILE* file_ptr,          // output SAT file
    logical mode_text,       // SAT or SAB mode
    BODY* body               // body to save
    );
```

Description: Refer to action.

save_entity_list_on_file

Function: SAT Save and Restore

Action: Saves an entity list to a file.

Prototype:
```
logical save_entity_list_on_file (
    FILE* file_ptr,          // output file pointer
    logical mode_text,       // TRUE is SAT
    ENTITY_LIST const& entities // entity list to
                             // save
    );
```

Description: The mode_text saves as SAT.

save_entity_on_file

Function: SAT Save and Restore

Action: Saves the general entity structure of the file, in text or binary.

Prototype:
```
logical save_entity_on_file (
    FILE* file_ptr,          // output file pointer
    logical mode_text,       // TRUE is SAT
    ENTITY* entity           // pointer to entity
    );
```

Description: Refer to action.

save_law

Function: SAT Save and Restore

Action: Saves a law to a .sat file.

Prototype:
```
void save_law (
    law* the_law             // law to save
    );
```

Description: For internal use only. Refer to the ENTITY class for details. Handles the save operation by writing out the savable data associated with a law.

set_file_info

Function: SAT Save and Restore

Action: Sets information about file.

Prototype:
```
void set_file_info (
    unsigned long,           // mask
    const FileInfo& info     // file information
    );
```

Description: Refer to action.

set_save_file_version

Function: SAT Save and Restore

Action: Sets the version number to be used for save file, for backwards compatibility.

Prototype:

```
void set_save_file_version (
    int save_maj            // major version number
        = 0,
    int save_min            // minor version number
        = -1
    );
```

Description: This method defaults to the current version. The default for the minor version number returns an error unless the major version number is 0.