Chapter 3.
SAT Save File Format

Topic: SAT Save and Restore

ACIS can store modeling information in external files, called *save files*. These files have an open format so that external applications, even those not based on ACIS, can have access to the ACIS geometric model.

The basic information needed to understand the ACIS file format (focusing on the reading, or *restore*, operation), includes the structure of the save file format, how data is encapsulated, the types of data written, and subtypes and references.

Save File Types

Topic: *SAT Save and Restore

ACIS supports two kinds of save files, SAT and SAB, which stand for “Standard ACIS Text” and “Standard ACIS Binary”, respectively. Although one is ASCII text and the other is binary data, the model data information stored in the two formats is identical, so the term *SAT file* is generally used to refer to either (when no distinction is needed).

SAT files are ASCII text files that may be viewed with a simple text editor. A SAT file contains carriage returns, white space and other formatting that makes it readable to the human eye. A SAT file has a .sat file extension.

SAB files cannot be viewed with a simple text editor and are meant for compactness and not for human readability. A SAB file has a .sab file extension. A SAB file uses delimiters between elements and binary tags, without additional formatting.

The binary formats supported are:

int . . . . . 4–byte 2s complement (as long)

long . . . . . 4–byte 2s complement

double . . . . . 8–byte IEEE

char . . . . . 1–byte ASCII

where “byte” is eight bits, and files are considered to be byte strings. For multi–byte data items, byte order normally just matches that of the processor being used, but a specific order may be imposed by compiling with the preprocessor macro BIG_ENDIAN or LITTLE_ENDIAN defined.

Structure of the Save File

Topic: SAT Save and Restore

A save file contains:

- Three-line header
- Entity records, representing the bulk of the data
- [optional] Marker for begin history data
- [optional] Old entity records needed for history and rollback
- [optional] Marker for end history data
- End marker

Beginning with ACIS Release 6.3, it is **required** that the product ID and units be populated for the file header (using class FileInfo) before you can save a SAT file. Refer to the reference templates for class FileInfo and function api_set_file_info for more information.

Summary Mode

Topic: SAT Save and Restore

This mode helps provide customers an ACIS option that allows them to save sat files in ”summary” mode. Summary mode allows sat files to be saved in a sort of ”compressed” fashion, thereby resulting in smaller sat files.

Usage

Topic: SAT Save and Restore

An ACIS option has been created, save_summary_mode, which is FALSE by default. When set to TRUE, sat files are saved in summary mode.

Saving in Summary Mode

Topic: SAT Save and Restore

When saving sat files in summary mode, three types of ”compression” occur:

- Indexing of IDs

- Summarization of logicals
- Summarization of enums

Indexing of IDs

This method of compression gives each ID an index, then allows that ID to be represented by it's index for the remainder of the sat file. For example, the first occurrence of 'loop' in a sat file would be represented by "loop%1". All subsequent occurrences of 'loop' would be represented by "%1".

Summarization of logicals

Some ACIS entities have sat–file data that has two–states, e.g., forward/reverse, out/in, no_rotate/rotate, etc. In summary mode, these states are simply represented as the single characters, T/F.

Summarization of enums

Some ACIS entities have sat–file data that has many states, e.g., non_singular/singular_low/singular_high/ singular_both, no_radius/single_radius/two_radii etc. In summary mode, these states are simply represented by E0/E1/E2/E3 and so on.

Restoring in Summary Mode

Topic: SAT Save and Restore

It is not necessary to enable summary mode to restore a summarized sat file. The restore mechanism automatically detects the presence of indexed id's and loads the file as a summarized sat file. Once ACIS determines that the sat file being restored is summarized, it assumes that the entire file is fully summarized and loads it as such. As such, it is not recommended that end users make modifications to summarized sat files.

In loading the summarized sat file, ACIS does not enable save_summary_mode. Enabling save_summary_mode can only be done through the option.

Compression Results

Topic: SAT Save and Restore

For sat files that are attribute heavy, summary mode can yield compressions of up to 40%. On average, the compression is 10%.

Reserved Characters in SAT Files

Topic: *SAT Save and Restore

Several characters have special meaning when found in "unknown entity" string data in ACIS SAT (.sat) files. Therefore, these characters should not be contained in any user string data written to SAT files. This includes string data in attributes, but applies to string data in any unknown entities.

Note *This only applies to unknown entity string data written to text save files (*.sat*); binary files (*.sab*) are not affected.*

The portion of the ACIS save (.sat) file restore code that processes unknown entities reserves the following special characters (these are discussed elsewhere in this chapter):

{ An opening (left) curly brace begins a subtype definition.

} A closing (right) curly brace terminates a subtype definition.

$ A dollar sign indicates a pointer definition.

A pound sign terminates an entity record.

@ An at sign starts a string record.

If these reserved characters are encountered in unknown entity string data when a .sat file is restored (read in), the unknown entity reader will not process the data correctly because these characters are interpreted as special tokens. Because this processing only applies to unknown entities, applications that "know about" entities containing these characters should be able to process the files. However, if the files are shared with other applications, problems may arise.

Save File Header

Topic: *SAT Save and Restore

The first record of the ACIS save file is a header, such as:

```
712 0 4 0
11 Scheme AIDE 11 ACIS 7.0 NT 24 Mon Apr 09 16:44:18 2001
–1 9.9999999999999995e–007 1e–010
```

Integer . . . . . An encoded version number. In the example, this is "712", which means the release is major release 7, minor release 1, and point release 2.

Integer . . . . . The total number of saved data records, or zero. If zero, then an end mark is required.

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

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

Integer . . . . . String length for the product string: "11".

String . . . . . ID for the product which produced the file: "Scheme AIDE".

Integer . . . . . String length for the ACIS version string: “11”.

String . . . . . ACIS version which produced the file: “ACIS 7.0 NT”. This may be different from the file version and can include the major release number, the minor release number, and the point release number. It also includes the platform on which it was produced.

Integer . . . . . String length for the date string: “24”.

String . . . . . Date file produced (in C ctime format): “Mon Apr 09 16:44:18 2001”.

Double . . . . . Number of millimeters represented by each unit in the model: “–1”.

Real . . . . . Value of SPAresabs when the file was produced: “9.9999999999999995e–007”.

Real . . . . . Value of SPAresnor when the file was produced: “1e–10”.

Version Numbers

Topic: *SAT Save and Restore

Spatial has always maintained the concept of a current version (release) number in ACIS, as well as a save version number. The save version allows one to create a SAT save file that can be read by a previous version of ACIS.

Beginning with ACIS Release 4.0, the SAT save file format does not change with minor releases, only with major releases. This allows applications that are based upon the same major version of ACIS to exchange data without being concerned about the save version. To provide this interoperability in a simple implementation, ACIS save files have contained a symbol that accurately identified the major version number, but not the minor version. This meant that applications created using the same major version of ACIS would produce compatible save files, regardless of their minor versions. This was accomplished by simply not incrementing the “internal” minor version number between major versions.

Beginning with Release 7.0, ACIS will again provide accurate major, minor, and point version numbers, which can be queried using the functions get_major_version, get_minor_version, and get_point_version. To summarize how release numbers and SAT changes are related:

- Major release: SAT file changes may be made; significant functionality changes likely; may require significant changes to existing applications
- Minor release: No SAT file changes are made; may provide new functionality; may require some minimal changes to existing applications
- Point release: Minor changes only (bug fixes)

Entity Records

Topic: *SAT Save and Restore

The header is followed by a sequence of entity records. Each entity record consists of a sequence number (optional), an entity type identifier, the entity data, and a terminator.

Pointers between entities are saved as integer index values, with NULL pointers represented by the value –1. ACIS pointer indices are preceded by $ in the SAT file, or by a binary Tag 12 in the SAB file.

Top level entities (e.g., body entities) are always the first records in the save file. The rest of the data records are in no particular order.

Beginning with ACIS Release 7.0, the format of ENTITY and HISTORY_STREAM records has been changed to accommodate the addition of entity IDs. A new integer field has been added to the entity record to hold the entity's ID. A value of –1 indicates that an ID has not yet been requested for the ENTITY. This is field #2 in any entity record, where the entity type is field #0. A new integer field has been added to history stream records (part of the history data section) to hold the next available entity ID in that stream. This is field #3 in the history stream record, where "history_stream" is the first field.

Optional Sequence Numbers

Topic: *SAT Save and Restore

The indexing of the entity records depends on the active ACIS options when the model was saved. If they are indexed, the indexing is sequential starting at 0.

```
–0 body $1 $2 $–1 $–1 #
    .
    .
    .
–25 point $–1 10 0 25 #
```

In this example from a SAT file, "–0" and "–25" are sequence numbers. In the first line, "$1 $2" happen to be pointers to records (not shown) with sequence numbers "–1" and "–2", respectively.

Even when the sequence numbers are not written to the file, they are implied by the order of the records in the file. Pointers to other records correspond to these implied sequence numbers. If sequence numbers are turned off, a record cannot be simply moved or removed from the save file, because this will create invalid index referencing when the file is restored.

If sequence numbers are turned on, an entity may be deleted by simply removing its record from the save file. Any references to the removed record's index become NULL pointers when the file is restored by ACIS. With sequence numbers on, records may also be rearranged within the file.

Save Identifier

Topic: *SAT Save and Restore

A save identifier is unique, reader-friendly string which has a one-to-one correspondence with an ACIS class. For example, “body” is a save identifier for the BODY class, which is derived from ENTITY.

ACIS classes perform most of the work associated with reading and writing save files. The restore methods of a given class can make references to the restore methods of other classes. Class names do not appear in the save file, but the save identifiers do. Both class names and save identifiers are contained within the index.

The save file sometimes strings save identifiers together using a dash (“–”). This often reflects the ACIS class derivation. For example, the save identifiers “plane–surface” have a class derivation consisting of ENTITY, SURFACE, and PLANE. The save identifiers “colour–tsl–attrib” have the class derivation ENTITY, ATTRIB, ATTRIB_TSL, and ATTRIB_COL.

Unique save identifier strings are required for private attributes that a developer may create. This is achieved by deriving a private base class from ATTRIB and giving it a unique name. Developers may use any class name for their private attributes, yet the full identifiers are unique in the save file.

If the save identifiers are not completely recognized by ACIS, a data structure from just the *recognized* classes is constructed and restored. The remaining data at the end of the record is remembered so that it is not lost by a later save. For unrecognized classes whose derivation is two or more levels removed from ENTITY, such as classes derived from CURVE, SURFACE, or ATTRIB, the minimum is to create a recognizable data structure so that references to the data structure are correct. For example, if a record from a derived class of ATTRIB is not recognized, an ATTRIB record is created in such a way that the chain of attributes remains connected for the entity owning the unrecognized attribute.

Character Sets Supported

Topic: *SAT Save and Restore

Beginning with ACIS Release 7.0, the SAT file will delimit strings using an @ sign. This allows SAT files to contain and support any character set, including Japanese or graphics characters. For backward compatibility, SAT files with strings that contain SAT delimiters will be modified when being saved in an earlier version. The SAT delimiters in the string will be replaced with underscores. This will insure that the file can be read in earlier versions of ACIS.

Entity Encapsulation in a Record

Topic: *SAT Save and Restore

The data for an entity is encapsulated in the order of its derivation from the basic ENTITY, from left to right. This is in the opposite order of the derivation of the identifier. The data for ENTITY is written first, followed by the data for the class directly derived from ENTITY, continuing down to the leaf class. For readability in the SAT file, each data field is separated by white space: a space, carriage return, or new line.

Because it is known that the encapsulation (and derivation) starts from ENTITY, the ENTITY save identifier is not written. In figure 3-1, this is shown in the middle by the empty double quotation marks, (“”).

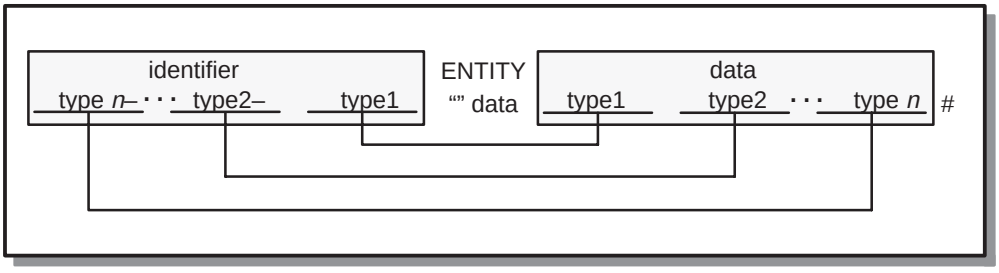

Figure 3-1. Format Entity Record

Starting from inside and going out, the ENTITY data is preceded by identifiers and is followed by the data for the identifiers. The identifiers correspond to classes derived directly from ENTITY. This type of encapsulation continues for all class derivations until the leaf class is reached.

In figure 3-1 for a SAT file, the class identifiers are separated from one another with dashes, while their data fields are separated with white space. The white space is typically a space, but may also be a new line. The last element of the record is the terminator character (#).

Terminator

Topic: *SAT Save and Restore

The pound sign (#) terminates the entity data in the SAT file. Tag 17 is the terminator in the SAB file. This allows unknown entities and attributes to be read and the start of the next entity to be located.

Subtypes and References

Topic: *SAT Save and Restore

Subtypes are frequently used in a save file in many of the entity records. Subtypes specify in more detail the characteristics of a geometry type. Interpolated curves (intcurve) and spline surfaces (spl_sur) are two of the major geometry types that make extensive use of subtypes.

Subtype definitions contain the bulk of the geometry information of a model and generally span numerous lines within a record of a SAT file. The subtype definition can be preceded and followed by other data pertinent to that record, and subtypes can be nested.

When a subtype definition is created as part of a record in the save file, it is numbered in an index table starting with 0. Because entity elements within a model often share geometry, the subtype index numbers can be referenced by other entity element records. When a particular record uses a subtype object which has been defined by another record, the latter record simply references the index number of the subtype rather than writing the entire set of data out again. These references are written out as "ref *n*". The ref indicates that this particular item has already been written out to the save file. The *n* stands for the subtype reference number, counting from the beginning of the file.

All subtype and reference designations are enclosed between subtype_start and subtype_end designators, which are curly braces "{ }" in a SAT file and Tag 15 and Tag 16 in a SAB file.

Geometry with Laws

Topic: *SAT Save and Restore

When a law is used to create geometry, as for curves, surfaces, a wire offsetting, and sweeping, the string denoting the law and any supporting data is saved to the save file.

A law is composed of one or more law symbol character strings. The law symbols are very similar to the adaptation of mathematical notation for use in computers. The valid syntax for the character strings is given in the law symbol templates. Laws support nesting of law symbols.

Laws are most likely to appear in the save file as part of a curve or surface subtype definition in the form of a lawintcur or a lawsplsur. Typically, the subtype definition starts with generic geometry information. Then the law specific information is presented. The law itself can reference and/or define any number of other model geometric elements and subtypes using law data definitions. Following the law definition and all associated law data in the save file, other common subtype information can appear.

End Marker

Topic: *SAT Save and Restore

The last entity record is followed by End-of-ACIS-data to mark the end of the ACIS save data.

```
End–of–ACIS–data
```

If the history save/restore option is turned on, the Begin–of–ACIS–History–Data and End–of–ACIS–History–Section markers together with old entities are listed before the End-of-ACIS-data marker.

History Markers

Topic: *SAT Save and Restore

When the history save/restore option is turned on, a new section is added to the save file before the End–of–ACIS–data marker. This new section comes immediately after the information pertaining to the entities of the active model.

```
Begin–of–ACIS–History–Data
```

The new history section starts with a marker Begin–of–ACIS–History–Data. Everything between this section marker and the End–of–ACIS–History–Section marker obeys the rules of history save.

This section of the save file lists entities which may no longer exist at the active state. However, these entities did exist at some point during the creation of the current model and are necessary for roll back and roll forward operations. Entities that were part of pruned branches are not saved.

When the history save/restore option is turned on, the End–of–ACIS–History–Section marker immediately follows the history data. Between this marker and the Begin–of–ACIS–data marker are more entity records. These adhere to the entity record structure.

```
End–of–ACIS–History–Section
```

The End–of–ACIS–History–Section marker is followed by the End–of–ACIS–data marker.

Class Restore Methods

Topic: SAT Save and Restore

Every ACIS class that can have information retrieved from a save file typically has a restore method. In most cases, this is either restore_common or restore_data.

When the restore methods are documented for a given class, they contain a function prototype, a description, and some pseudo-code to describe the actual data that is retrieved from the SAT file. The pseudo-code can include references to other class restore functions or to other generic functions to handle known data types in the save file.

For example, when restoring a cone entity from a save file, the cone::restore_data method references the ellipse::restore_data method as well as the surface::restore_data method. It also references common input functions like read_real and read_logical. Likewise, the ellipse::restore_data method references the common input functions read_position, read_unit_vector, and read_vector.

Hence, finding out exactly all of the data associated with a given save identifier may involve tracing through several classes and common input functions.

Tracing SAT Data

Topic: SAT Save and Restore

Using the ACIS online help “index search can speed the interpretation of the information in a SAT file. The index search lists both class names and save identifiers.

1. Locate the index entry for the save identifier of interest, for example, the class BODY. Follow the index entry to the class information, of which the save identifier is a part.
2. The first three lines of the SAT file contain header information. Parse these lines for information of value, such as the modeling units. For more details about the header, refer to the section *Save File Header.*
3. For an entity record within the SAT file, read the inner-most save identifier.
    a. For more details about entity records, refer to the section *Entity Records*.
    b. For more details about the save identifiers, refer to the section *Save Identifiers*.
    c. For more details about finding the inner–most save identifiers, refer to the section *Entity Encapsulation in a Record*.
4. Locate the index entry for the save identifier.
5. Follow the index entry for the save identifier to its owning class description in the documentation.
6. The restore methods of the class specify the data associated with that save identifier. This data begins just to the right of the inner-most save identifier of the SAT record. For more details about restore methods, refer to the section *Class Restore Methods*.
7. The data associated with the save identifier is represented by the pseudo code of the restore methods.
8. The restore methods may reference restore methods for other classes. If required, go to those classes and follow their restore methods. Be sure to keep track of the class restore stack so that tracing can pop back to the correct class restore method.
9. When finished with the data for the given save identifier, go to step 4. for the next inner-most save identifier, if applicable.
10. If there are no more save identifiers and the data portion has encountered the entity record terminator (“#”), go to step 3. for the next entity record.

For example, assume the following line was found in a SAT file.

```
cone-surface $-1 0 0 0 0 0 1 10 0 0 1 I I 0 1 forward I I I I #
```

The “surface” keyword is the inner-most save identifier which is next to the NULL ENTITY. pointer ($–1). The documentation index for the word “surface” leads to the SURFACE class. So, this is the place to start tracing.

SURFACE Class

```
public: void SURFACE::restore_common ();

No data                  This class does not save any data
```

In this case, the restore method of the SURFACE class does not read any data.

CONE Class

The “cone” keyword is the next inner-most save identifier. The documentation index for the word “cone” goes to the CONE class.

```
public: void CONE::restore_common ();

cone::restore_data       Cone data definition.
```

The restore method of the CONE class references the restore_data method of the cone class.

cone Class

```
public: void cone::restore_data ();

ellipse::restore_data   Restore the information for the base ellipse

read_real   Sine of cone angle

read_real   Cosine of cone angle

if (restore_version_number < CONE_SCALING_VERSION)

    // the u parameter scale is obtained from the ellipse major axis

else

    read_real           u parameter scale

if (restore_version_number < SURFACE_VERSION)

    // the reverse u flag is set to FALSE

else

    read_logical        u parameter reversed, either “forward” or reversed”

surface::restore_data   Generic surface data
```

The restore method of the cone class references the restore_data method of the ellipse class.

ellipse Class

```
public: void ellipse::restore_data ();
```

```
read_position           Position of the center of the ellipse.
read_unit_vector        Unit vector that is normal to plane of the ellipse.
read_vector             Major axis of the ellipse.
read_real               Ratio of the radii.
curve::restore_data     Restore the underlying curve of the ellipse.
```

The restore method of the ellipse class finally calls some input functions to retrieve information from the SAT file. Specifically, it reads the position of the base ellipse, which is the first “0 0 0” after the “$–1”. It reads a unit vector for the base normal, which is the next “0 0 1”. It reads a vector for the major axis of the ellipse, which is “10 0 0”. It reads a real, “1”, for the ratio of the ellipse major to minor axis. Then it accesses the restore method of the curve class.

curve Class

```
public: void curve::restore_data ();
if (restore_version_number >= BNDCUR_VERSION)
    read_interval       Interval for the subset range.
```

The restore method of the curve class reads in an interval. The interval itself is made up of two logicals, which happen to be the next “I I”, for two infinite values.

The curve::restore_data method returns to ellipse::restore_data, which then returns to cone::restore_data so that processing can resume. The next two values, “0 1”, represent the sine and cosine of the cone angle, respectively. The *u* parameter scale is obtained from the ellipse’s major axis. It reads the logical “forward” before accessing the restore method of the surface class.

surface Class

```
public: void surface::restore_data ();
if (restore_version_number >= BNDSUR_VERSION)
    read_interval       subset u interval
    read_interval       subset v interval
```

The surface::restore_data method reads in two intervals which in this case are infinite, “I I I I”.

There are no other save identifiers to parse and the “#” indicates that there is no more data associated with that record.

Save File Example

Topic: *SAT Save and Restore

The following simple example shows a SAT file with topology and geometry. The first three lines are the header, followed by the entity records, and finally the end marker. Optional sequence numbers were included.

Record lines “–0” and “–1” are explained in more detail following the full example to show exactly what each item is and where to locate that information in this manual.

```
# ACIS Scheme Commands

(option:set
    ”sequence_save_files” #t)
(solid:cylinder
    (position 0 0 0)
    (position 0 20 0)
    10)
(part:save ”exp_cyl”)
```

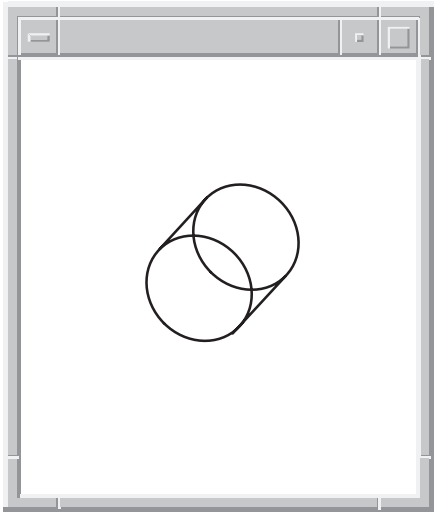

Figure 3-2. Save File Example

```
400 0 1 0
11 Scheme AIDE 11 ACIS 4.0 NT 24 Mon Apr 12 13:59:03 1998
25.4 1e-06 1e-10
-0 body $1 $2 $-1 $3 #
-1 display_attribute-st-attrib $-1 $4 $-1 $0 1 #
-2 lump $-1 $-1 $5 $0 #
-3 transform $-1 1 0 0 0 0 -1 0 1 0 0 10 0 1 rotate no_reflect
   no_shear #
-4 rgb_color-st-attrib $-1 $6 $1 $0 0 1 0 #
-5 shell $-1 $-1 $-1 $7 $-1 $2 #
-6 id_attribute-st-attrib $-1 $-1 $4 $0 1 #
-7 face $-1 $8 $9 $5 $-1 $10 forward single #
-8 face $-1 $11 $12 $5 $-1 $13 forward single #
-9 loop $-1 $14 $15 $7 #
-10 cone-surface $-1 0 0 0 0 0 1 10 0 0 1 I I 0 1 forward I I I I #
-11 face $-1 $-1 $16 $5 $-1 $17 forward single #
-12 loop $-1 $-1 $18 $8 #
-13 plane-surface $-1 0 0 -10 0 0 -1 -1 0 0 forward_v I I I I #
-14 loop $-1 $-1 $19 $7 #
-15 coedge $-1 $15 $15 $18 $20 1 $9 $-1 #
-16 loop $-1 $-1 $21 $11 #
-17 plane-surface $-1 0 0 10 0 0 1 1 0 0 forward_v I I I I #
-18 coedge $-1 $18 $18 $15 $20 0 $12 $-1 #
-19 coedge $-1 $19 $19 $21 $22 1 $14 $-1 #
-20 edge $-1 $23 $23 $18 $24 forward #

-21 coedge $-1 $21 $21 $19 $22 0 $16 $-1 #
-22 edge $-1 $25 $25 $21 $26 forward #
-23 vertex $-1 $20 $27 #
-24 ellipse-curve $-1 0 0 -10 0 0 -1 10 0 0 1 I I #
-25 vertex $-1 $22 $28 #
-26 ellipse-curve $-1 0 0 10 0 0 1 10 0 0 1 I I #
-27 point $-1 10 0 -10 #
-28 point $-1 10 0 10 #
End-of-ACIS-data
```

The first three lines of the file are the header. The fourth line, shown below, describes a body entity.

```
-0 body $1 $2 $-1 $3 #
```

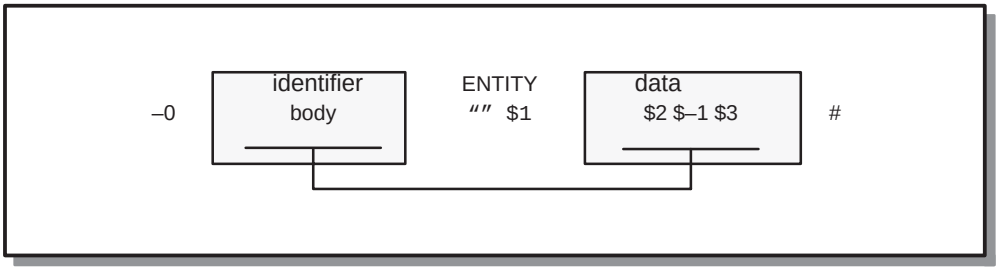

Figure 3-3. Body Entity

Code Output	Class	Description
–0		Sequence number 0; first data record of file.
body	**BODY**	ident – (BODY derived from ENTITY)
$1	**ENTITY**	ENTITY data – Pointer to its attribute
$2	**BODY**	BODY data – Pointer to body’s lump
$–1	**BODY**	BODY data – Pointer to body’s wire
$3	**BODY**	BODY data – Pointer to body’s transform
#		Terminator

The fifth line of the file, shown below, describes a display attribute entity.

```
–1 display_attribute–st–attrib $–1 $4 $–1 $0 1 #
```

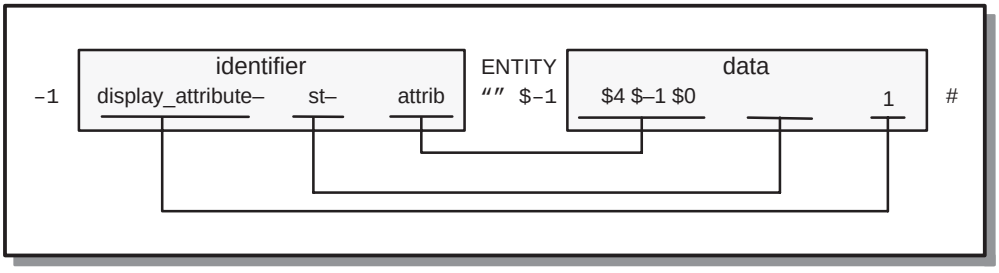

Figure 3-4. Display Attribute Entity

Code Output	Class	Description
–1	**ENTITY** (implied)	Sequence number 1; address for attribute pointer called out in sequence number 0. The double quotation marks (“”) are meant to signify that ENTITY is implied.
display_attribute–	**DISPLAY_ATTRIB**	save identifier for the DISPLAY_ATTRIB
st–	**ATTRIB_ST**	save identifier for the ATTRIB_ST class
attrib	**ATTRIBUTE**	save identifier for the ATTRIBUTE class
$–1	**ENTITY**	ENTITY data – Attribute $rec_num
$4	**ATTRIBUTE**	Pointer to the next attribute
$–1	**ATTRIBUTE**	Pointer to the previous attribute
$0	**ATTRIBUTE**	Pointer to owner
1	**DISPLAY_ATTRIB**	Display revision
#		Terminator

Subtypes and References Example

Topic: *SAT Save and Restore

As a more graphical illustration of how subtyping and referencing work in a save file, several Scheme commands and a resulting SAT file are presented. The Scheme commands generate a cylinder which has one end bounded by a spline face. Another cylinder intersects the cylinder along the spline face, forming a cylindrical groove in the spline face. The spline surface is used in the definition of several spline curves in this example.

```
# Scheme AIDE Commands

(set:options
    ”sequence_save_files” #t)
(solid:cylinder
    (position 0 0 0)
    (position 0 0 20)
    10)
(solid:wiggle 20 20 40 1 1 1 1)
(solid:intersect
    (entity 1)(entity 2))
(solid:cylinder
    (position –20 0 20)
    (position 20 0 20)
    3)
(solid:subtract
    (entity 1)(entity 3))
(part:save ”exm_ref.sat”)
```

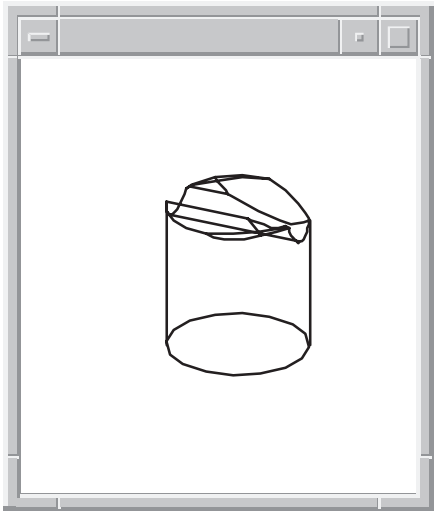

Figure 3-5. Creating a SAT File

The following example is taken from the file exm_ref.sat created in the above Scheme code. It does not list all sequence numbers from 0 to 171, but rather only a portion of them. The highlighted entries for sequence numbers 17, 96, and 118 are explained in more detail following this listing.

```
400 0 1 0
11 Scheme AIDE 11 ACIS 4.0 NT 24 Mon Apr 12 13:59:03 1998
25.4 1e–06 1e–10
–0 body $1 $2 $–1 $3 #
–1 display_attribute–st–attrib $–1 $4 $–1 $0 3 #
–2 lump $–1 $–1 $5 $0 #
–3 transform $–1 1 0 0 0 1 0 0 0 1 0 0 10 1
       rotate no_reflect no_shear #
–4 rgb_color–st–attrib $–1 $6 $1 $0 0 1 0 #
```

```
-5 shell $-1 $-1 $-1 $7 $-1 $2 #
-6 id_attribute-st-attrib $-1 $-1 $4 $0 35 #
-7 face $-1 $8 $9 $5 $-1 $10 reversed single #
-8 face $-1 $11 $12 $5 $-1 $13 forward single #
-9 loop $-1 $-1 $14 $7 #
-10 cone-surface $-1 0 0 10 1 0 0 0 0 -3 1 I I 0 1 forward I I I I
#
-11 face $-1 $15 $16 $5 $-1 $17 forward single #
-12 loop $-1 $-1 $18 $8 #
-13 plane-surface $-1 0 0 10 0 0 1 1 0 0 forward_v I I I I #
-14 coedge $-1 $19 $20 $21 $22 1 $9 $-1 #
-15 face $-1 $23 $24 $5 $-1 $25 forward single #
-16 loop $-1 $-1 $26 $11 #
-17 spline-surface $-1 forward { exactsur nubs 3 3 open
       open none none 3 2
    .
    .   [Data not shown; this is subtype number 0]
    .
    } I I I I #
-18 coedge $-1 $27 $28 $29 $30 0 $12 $-1 #
    .
    .   [Sequence entries 19 through 94 not shown]
    .
-95 vertex $-1 $93 $138 #
-96 intcurve-curve $-1 forward { surfintcur nubs 3 open 4
    .
    .   [Data not shown; this is subtype number 8]
    .
    spline forward { exactsur nubs 3 3 open open none none 3 3
    .
    .   [Data not shown; this is subtype number 9]
    .
    } I I I I
    plane 0 0 10 0 0 1 1 0 0 forward_v I I I I
    .
    .   [Data not shown]
    .
    nullbs
    F 0 F 4.1671539659088168 } I I #
-97 coedge $-1 $64 $54 $134 $139 1 $65 $-1 #
    .
    .   [Sequence entries 98 through 116 not shown]
    .
-117 vertex $-1 $73 $152 #
-118 intcurve-curve $-1 forward { surfintcur nubs 3 open 4
```

```
    .
    .   [Data not shown; this is subtype number 10
    .   and makes a reference to subtype number 8]
    spline forward { ref 8 } I I I I
    .
    .   [Data not shown]
    .
    F 3.2425629808368615 F 7.4005854616279674 } I I #
–119 vertex $–1 $115 $153 #
    .
    .   [Sequence entries 120 through 168 not shown]
    .
–169 vertex $–1 $164 $171 #
–170 ellipse–curve $–1 0 0 –10 0 0 –1 10 0 0 1 I I #
–171 point $–1 0 10 –10 #
End–of–ACIS–data
```

The first occurrence of a subtype definition for exactsur is on sequence number 17, and is numbered 0 in the subtype index table. This is explained in the following table.

Table 3-1. Code Output Description

Code Output	Class	Description
17		Sequence number 17.
spline–	**SPLINE**	save identifier for the spline class
surface	**SURFACE**	save identifier for the SURFACE class
$–1	**ENTITY**	Pointer to the attribute record
forward	**spline**	Parameter defining the sense direction for the spline curve.
{		Start of the subtype definition. In this particular case, the subtype index number is 0.
exactsur	**exact_spl_sur**	Subtype name is exactsur and uses data element spl_sur.

Code Output	Class	Description
	spl_sur	Data element, which has a bs3 surface and a real used as the fit tolerance.
nubs	**bs3_surface**	Ident: used as part of B-spline curve definition.
3	**bs3_surface**	integer, *u*-degree.
3	**bs3_surface**	integer, *v*-degree.
open	**bs3_surface**	ident: *u*-closure identifier.
open	**bs3_surface**	ident: *v*-closure identifier.
none	**bs3_surface**	ident: *u*-singularity.
none	**bs3_surface**	ident: *v*-singularity.
3	**bs3_surface**	ident: number of *u*-knots.
2	**bs3_surface**	ident: number of *v*-knots.
[Data not shown]		Data that was left out related to the *u*-knot values, their paired multiplicity, the *v*-knot values, their paired multiplicity, associated *x, y, z* coordinates, and a fit tolerance.
}		End of the subtype definition. In this particular case, the subtype index number is 0.
I I	**surface**	interval – *u*-parameter range.
I I	**surface**	interval – *v*-parameter range.
#		Terminator for sequence number –17.

The listing before the above table also has an example of a reference. Sequence number –118 has the text "`{ ref 8 }`" embedded within its information. This is a direct reference to subtype object 8. (This references the ninth subtype definition within the file, because the subtype index numbering begins with 0.)

The subtype object with the index number 8 is defined on sequence number 96. Thus, that definition of a surfintcur is used as part of sequence number 118. Note that the reference to subtype 8 in sequence number 118 actually occurs during the definition of another subtype. Moreover, nested subtype definitions are also supported, as is seen within the subtype definition on sequence number 96.

Constant Definitions and #define

Topic: *SAT Save and Restore

Constant definitions were often used during the evaluation of the data stored for a given save identifier. These are declared in #define statements in the file versions.hxx. This poses no problems for someone having ACIS. However, for those who do not have ACIS, the header declarations are listed below.

The versions.hxx file contains a list of the ACIS version numbers where the format of save files changed. It is used in individual ENTITY (and other) "restore" routines to allow old save files to be restored into new ACIS programs.

```
// This file contains a list of the version numbers of ACIS
// at which the format of ″save″ files has changed. It is used
// in individual ENTITY (and other) ″restore″ routines to allow
// old save files to be restored into new ACIS programs.

// First declare the variables which contain the version numbers
// of the current save file being restored. There is a major and
// minor component, and a portmanteau number which combines the
// two, and is the value actually placed in the save file.

#if !defined( VERSIONS_HDR_DEF )
#define  VERSIONS_HDR_DEF

#include ″kernel/dcl_kern.h″

extern DECL_KERN int& get_restore_major_version();
extern DECL_KERN int& get_restore_minor_version();
// Combined version number.
extern DECL_KERN int& get_restore_version_number();

#ifndef restore_version_number
#define restore_version_number get_restore_version_number()
#endif
```

```
// Now the same for the current ”save” file. This is normally set
// to the version number of this version of ACIS, but can be set
// backwards by the application to cause backwards–compatible save
// files to be produced (provided the data structure is itself
// backwards–compatible).

extern DECL_KERN int& get_save_major_version();
extern DECL_KERN int& get_save_minor_version();
extern DECL_KERN int& get_save_version_number(); // Combined version number.

#ifndef save_version_number
#define save_version_number get_save_version_number()
#endif

// Macros to define the way the portmanteau version number is
// obtained from the major and minor versions.

#define PORTMANTEAU( maj, min ) (100 * maj + min)
#define MAJOR_VERSION( port ) (port / 100)
#define MINOR_VERSION( port ) (port % 100)

// Release version numbers.

// The version at which the start and end parameter values of
// edges were removed from the save file, and recomputed when
// required.

const int PARAM_MAJOR = 1;
const int PARAM_MINOR = 1;
const int PARAM_VERSION = PORTMANTEAU( PARAM_MAJOR, PARAM_MINOR );

// The version at which LUMPs were introduced, to subdivide bodies
// into disjoint connected regions.

const int LUMP_MAJOR = 1;
const int LUMP_MINOR = 1;
const int LUMP_VERSION = PORTMANTEAU( LUMP_MAJOR, LUMP_MINOR );

// The version at which curve types were first output to the save
// file as character strings instead of integers.

const int CURVE_MAJOR = 1;
const int CURVE_MINOR = 3;
const int CURVE_VERSION = PORTMANTEAU( CURVE_MAJOR, CURVE_MINOR );

// The version at which surface types were first output to the save
// file as character strings instead of integers.

const int SURFACE_MAJOR = 1;
const int SURFACE_MINOR = 3;
const int SURFACE_VERSION =
    PORTMANTEAU( SURFACE_MAJOR, SURFACE_MINOR );
```

```
// The version at which subtypes of intcurves were defined requiring
// classification in the save file.

const int INTCURVE_MAJOR = 1;
const int INTCURVE_MINOR = 3;
const int INTCURVE_VERSION =
    PORTMANTEAU( INTCURVE_MAJOR, INTCURVE_MINOR );


// The version at which subtypes of splines were defined requiring
// classification in the save file.

const int SPLINE_MAJOR = 1;
const int SPLINE_MINOR = 3;
const int SPLINE_VERSION = PORTMANTEAU( SPLINE_MAJOR, SPLINE_MINOR );


// The version at which pointers were first distinguished from integers
// in text files by a dollar sign.

const int DOLLAR_MAJOR = 1;
const int DOLLAR_MINOR = 3;
const int DOLLAR_VERSION = PORTMANTEAU( DOLLAR_MAJOR, DOLLAR_MINOR );


// The version at which sharable objects like int_cur and spl_sur
// were first surrounded by curly braces, and could be shared in the
// save file.

const int SHARABLE_MAJOR = 1;
const int SHARABLE_MINOR = 4;
const int SHARABLE_VERSION = PORTMANTEAU( SHARABLE_MAJOR, SHARABLE_MINOR );


// The version at which blend attributes lost their "form" integer
// value (which was always zero, and so redundant).

const int BLEND_MAJOR = 1;
const int BLEND_MINOR = 5;
const int BLEND_VERSION = PORTMANTEAU( BLEND_MAJOR, BLEND_MINOR );


// The version at which parameter curves may have the defining
// parameter curve and surface in either slot 1 or 2.

const int PARCUR_MAJOR = 1;
const int PARCUR_MINOR = 5;
const int PARCUR_VERSION = PORTMANTEAU( PARCUR_MAJOR, PARCUR_MINOR );


// The version at which pcurves may have different subtypes, and so
// need a type code.

const int PCURVE_MAJOR = 1;
const int PCURVE_MINOR = 5;
const int PCURVE_VERSION = PORTMANTEAU( PCURVE_MAJOR, PCURVE_MINOR );
```

```
// The version at which faces may be double-sided, either free sheets
// or embedded in material.

const int TWOSIDE_MAJOR = 1;
const int TWOSIDE_MINOR = 5;
const int TWOSIDE_VERSION = PORTMANTEAU( TWOSIDE_MAJOR, TWOSIDE_MINOR );


// The version at which logical values are written to the save file
// as "T" or "F" instead of as integers.

const int LOGIO_MAJOR = 1;
const int LOGIO_MINOR = 5;
const int LOGIO_VERSION = PORTMANTEAU( LOGIO_MAJOR, LOGIO_MINOR );


// The version at which coedges were first expected to be sorted clockwise
// around the edge.

const int SORTCOED_MAJOR = 1;
const int SORTCOED_MINOR = 5;
const int SORTCOED_VERSION = PORTMANTEAU( SORTCOED_MAJOR, SORTCOED_MINOR );


// The version at which intervals could be (semi-)infinite, and so require
// different save file format.

const int INFINT_MAJOR = 1;
const int INFINT_MINOR = 6;
const int INFINT_VERSION = PORTMANTEAU( INFINT_MAJOR, INFINT_MINOR );


// The version at which curves have a subset range to bound them within
// their natural range.

const int BNDCUR_MAJOR = 1;
const int BNDCUR_MINOR = 6;
const int BNDCUR_VERSION = PORTMANTEAU( BNDCUR_MAJOR, BNDCUR_MINOR );


// The version at which surfaces have a subset range to bound them within
// their natural range.

const int BNDSUR_MAJOR = 1;
const int BNDSUR_MINOR = 6;
const int BNDSUR_VERSION = PORTMANTEAU( BNDSUR_MAJOR, BNDSUR_MINOR );


// The version at which multiple entities can be saved into a single save
// file unit (as opposed to multiple self-contained sections of a file).

const int MULTSAV_MAJOR = 1;
const int MULTSAV_MINOR = 5;
const int MULTSAV_VERSION = PORTMANTEAU( MULTSAV_MAJOR, MULTSAV_MINOR );
```

```
// The version in which save and restore started using a FileInterface
// object to do the I/O.

const int FILEINTERFACE_MAJOR = 1;
const int FILEINTERFACE_MINOR = 6;
const int FILEINTERFACE_VERSION =
    PORTMANTEAU( FILEINTERFACE_MAJOR, FILEINTERFACE_MINOR );


// The version in which "Wire Booleans" were implemented, requiring
// extensions to the data structure.

const int WIREBOOL_MAJOR = 1;
const int WIREBOOL_MINOR = 7;
const int WIREBOOL_VERSION = PORTMANTEAU( WIREBOOL_MAJOR, WIREBOOL_MINOR );


// The version in which "3D eye refinements" were implemented.

const int THREEDEYE_REF_MAJOR = 1;
const int THREEDEYE_REF_MINOR = 7;
const int THREEDEYE_REF_VERSION =
    PORTMANTEAU(THREEDEYE_REF_MAJOR,THREEDEYE_REF_MINOR);


// STI jmb: Version in which the History Manager was introduced

const int HISTORY_MAJOR = 1;
const int HISTORY_MINOR = 7;
const int HISTORY_VERSION =
    PORTMANTEAU( HISTORY_MAJOR, HISTORY_MINOR);
// STI jmb: end


// The version in which "Safe Ranges" for intcurves were implemented,
// requiring an extra field in the data structure.

const int SAFERANGE_MAJOR = 1;
const int SAFERANGE_MINOR = 7;
const int SAFERANGE_VERSION = PORTMANTEAU( SAFERANGE_MAJOR, SAFERANGE_MINOR );


// The version which introduced angled cross curves at the ends of
// face-face blends, requiring extra fields in ATTRIB_FFBLEND.

const int ANG_XCUR_MAJOR = 1;
const int ANG_XCUR_MINOR = 7;
const int ANG_XCUR_VERSION = PORTMANTEAU( ANG_XCUR_MAJOR, ANG_XCUR_MINOR );


// The version at which advanced blending facilities were first made
// available.

const int ADV_BL_MAJOR = 1;
const int ADV_BL_MINOR = 8;
const int ADV_BL_VERSION = PORTMANTEAU( ADV_BL_MAJOR, ADV_BL_MINOR );
```

```
// STI dgh – begin
// The version where the Intergraph spline library was first introduced.

//const int IGRSPLINE_MAJOR = 2;
//const int IGRSPLINE_MINOR = 0;
//const int IGRSPLINE_VERSION = PORTMANTEAU( IGRSPLINE_MAJOR, IGRSPLINE_MINOR
);
// STI dgh – end


// The version where the save/restore of logicals, enums was made consistent

const int CONSISTENT_MAJOR = 2;
const int CONSISTENT_MINOR = 0;
const int CONSISTENT_VERSION = PORTMANTEAU(CONSISTENT_MAJOR,
CONSISTENT_MINOR);


// The version where the additional header information was added

const int FILEINFO_MAJOR = 2;
const int FILEINFO_MINOR = 0;
const int FILEINFO_VERSION = PORTMANTEAU(FILEINFO_MAJOR, FILEINFO_MINOR);


// The version where the mesh classes were added

const int MESH_MAJOR = 2;
const int MESH_MINOR = 0;
const int MESH_VERSION = PORTMANTEAU(MESH_MAJOR, MESH_MINOR);


// The version where extended curves and surfaces were introduced.

const int EXT_CU_SF_MAJOR = 2;
const int EXT_CU_SF_MINOR = 1;
const int EXT_CU_SF_VERSION = PORTMANTEAU(EXT_CU_SF_MAJOR, EXT_CU_SF_MINOR);


// The version where coedges were given sense enumeration strings.

const int COEDGE_SENSE_MAJOR = 2;
const int COEDGE_SENSE_MINOR = 2;
const int COEDGE_SENSE_VERSION =
    PORTMANTEAU(COEDGE_SENSE_MAJOR, COEDGE_SENSE_MINOR);


// The version at which Skin/Loft surfaces have a subset range to bound them
// within their natural range.

const int ARCWISE_SKIN_MAJOR = 2;
const int ARCWISE_SKIN_MINOR = 2;
const int ARCWISE_SKIN_VERSION =
    PORTMANTEAU( ARCWISE_SKIN_MAJOR, ARCWISE_SKIN_MINOR );
```

```
// The version in which adv_var_blend attributes output a logical
// specifying if two radii functions are used

const int ADV_VAR_BLEND_TWO_RADII_MAJOR = 2;
const int ADV_VAR_BLEND_TWO_RADII_MINOR = 2;
const int ADV_VAR_BLEND_TWO_RADII_VERSION =
    PORTMANTEAU( ADV_VAR_BLEND_TWO_RADII_MAJOR,
ADV_VAR_BLEND_TWO_RADII_MINOR);

// The version in which laws first where added to ACIS

const int LAW_MAJOR = 2;
const int LAW_MINOR = 2;
const int LAW_VERSION = PORTMANTEAU( LAW_MAJOR, LAW_MINOR );

// The version in which reflection of offset_spl_surs was handled

const int OFFSET_REV_MAJOR = 2;
const int OFFSET_REV_MINOR = 2;
const int OFFSET_REV_VERSION = PORTMANTEAU( OFFSET_REV_MAJOR,
OFFSET_REV_MINOR);

// The version in which discontinuity information was stored in int_curs and
// spl_surs

const int DISCONTINUITY_MAJOR = 3;
const int DISCONTINUITY_MINOR = 0;
const int DISCONTINUITY_VERSION =
    PORTMANTEAU( DISCONTINUITY_MAJOR, DISCONTINUITY_MINOR);

// The version in which taper_spl_surs came into existence

const int TAPER_EXISTENCE_MAJOR = 2;
const int TAPER_EXISTENCE_MINOR = 1;
const int TAPER_EXISTENCE_VERSION =
    PORTMANTEAU( TAPER_EXISTENCE_MAJOR, TAPER_EXISTENCE_MINOR );

// The version in which taper_spl_surs were split into derived classes

const int TAPER_MAJOR = 3;
const int TAPER_MINOR = 0;
const int TAPER_VERSION =
    PORTMANTEAU( TAPER_MAJOR, TAPER_MINOR);

// The version in which net surface was added to ACIS

const int NET_SPL_MAJOR = 3;
const int NET_SPL_MINOR = 0;
const int NET_SPL_VERSION = PORTMANTEAU( NET_SPL_MAJOR, NET_SPL_MINOR );

// The version in which law curves and surfaces where added to ACIS

const int LAW_SPL_MAJOR = 4;
const int LAW_SPL_MINOR = 0;
const int LAW_SPL_VERSION = PORTMANTEAU( LAW_SPL_MAJOR, LAW_SPL_MINOR );
```

```
// The version at which cones have a new member, representing the
// scaling factor of the u parameter.

const int CONE_SCALING_MAJOR = 4;
const int CONE_SCALING_MINOR = 0;
const int CONE_SCALING_VERSION =
    PORTMANTEAU( CONE_SCALING_MAJOR, CONE_SCALING_MINOR );

// The version in which laws in lofts where added to ACIS

const int LOFT_LAW_MAJOR = 4;
const int LOFT_LAW_MINOR = 0;
const int LOFT_LAW_VERSION = PORTMANTEAU( LOFT_LAW_MAJOR, LOFT_LAW_MINOR );

// The version in which refinement ”min_u_grid_lines and min_v_grid_lines”
// were added.

const int REF_MIN_UV_GRID_MAJOR = 4;
const int REF_MIN_UV_GRID_MINOR = 0;
const int REF_MIN_UV_GRID_VERSION =
    PORTMANTEAU(REF_MIN_UV_GRID_MAJOR,REF_MIN_UV_GRID_MINOR);

// The version at which vertex blend attributes were given a new member
// describing the method of auto setback calculation, if any.

const int VBLEND_AUTO_MAJOR = 4;
const int VBLEND_AUTO_MINOR = 0;
const int VBLEND_AUTO_VERSION =
    PORTMANTEAU(VBLEND_AUTO_MAJOR,VBLEND_AUTO_MINOR);

// The version at which the true rolling–ball envelope surface was introduced
// for var_blend_spl_sur’s.

const int BL_ENV_SF_MAJOR = 4;
const int BL_ENV_SF_MINOR = 0;
const int BL_ENV_SF_VERSION =
    PORTMANTEAU(BL_ENV_SF_MAJOR,BL_ENV_SF_MINOR);

// The version in which ellipse offset with an ellipse base curve so that they
// will extend the ellipse and not linearly.

const int ELLIPSE_OFFSET_MAJOR = 5;
const int ELLIPSE_OFFSET_MINOR = 0;
const int ELLIPSE_OFFSET_VERSION = PORTMANTEAU( ELLIPSE_OFFSET_MAJOR,
ELLIPSE_OFFSET_MINOR );

// The version at which Tolerant Modeling was introduced.

const int TOL_MODELING_MAJOR = 5;
const int TOL_MODELING_MINOR = 0;
const int TOL_MODELING_VERSION =
    PORTMANTEAU(TOL_MODELING_MAJOR,TOL_MODELING_MINOR);
```

```
// The version in which approximating geometry could be stored in summary
// form.

const int APPROX_SUMMARY_MAJOR = 5;
const int APPROX_SUMMARY_MINOR = 0;
const int APPROX_SUMMARY_VERSION =
    PORTMANTEAU( APPROX_SUMMARY_MAJOR, APPROX_SUMMARY_MINOR );

// The version at which a scaling factor was saved for tapers.

const int TAPER_SCALING_MAJOR = 5;
const int TAPER_SCALING_MINOR = 0;
const int TAPER_SCALING_VERSION =
    PORTMANTEAU(TAPER_SCALING_MAJOR,TAPER_SCALING_MINOR);

// The version at which a scaling factor was saved for tapers.

const int LAZY_B_SPLINE_MAJOR = 5;
const int LAZY_B_SPLINE_MINOR = 0;
const int LAZY_B_SPLINE_VERSION =
    PORTMANTEAU(LAZY_B_SPLINE_MAJOR,LAZY_B_SPLINE_MINOR);

// The version at which multi-surfaces were introduced to Deformable Modeling

const int DM_MULTI_SURF_MAJOR = 5;
const int DM_MULTI_SURF_MINOR = 0;
const int DM_MULTI_SURF_VERSION =
    PORTMANTEAU(DM_MULTI_SURF_MAJOR,DM_MULTI_SURF_MINOR);

// The version at which the copy_owner method was added to ATTRIB_GEN_NAME.

const int GA_COPY_ACTION_MAJOR = 6;
const int GA_COPY_ACTION_MINOR = 0;
const int GA_COPY_ACTION_VERSION =
    PORTMANTEAU(GA_COPY_ACTION_MAJOR,GA_COPY_ACTION_MINOR);

// The version at which link constraint color persistence was introduced to
// Deformable Modeling

const int DM_MULTI_SURF_COLOR_MAJOR = 6;
const int DM_MULTI_SURF_COLOR_MINOR = 0;
const int DM_MULTI_SURF_COLOR_VERSION =
    PORTMANTEAU(DM_MULTI_SURF_COLOR_MAJOR,DM_MULTI_SURF_COLOR_MINOR);

// The version at which Skin/Loft surfaces error should be recalculated
// so that the intersector will not box out valid skin intersections.

const int RECAL_SKIN_ERROR_MAJOR = 5;
const int RECAL_SKIN_ERROR_MINOR = 2;
const int RECAL_SKIN_ERROR_VERSION =
    PORTMANTEAU( RECAL_SKIN_ERROR_MAJOR, RECAL_SKIN_ERROR_MINOR );
```

```
// The version at which the fact whether the ruled taper was
// ruled in u or in v was saved for tapers.

const int TAPER_U_RULED_MAJOR = 6;
const int TAPER_U_RULED_MINOR = 0;
const int TAPER_U_RULED_VERSION =
    PORTMANTEAU(TAPER_U_RULED_MAJOR,TAPER_U_RULED_MINOR);

// Multiple changes to DM SAT file in 6.0
// including point tangent constraints
// and saving of default shapes

const int DM_60_MAJOR = 6;
const int DM_60_MINOR = 0;
const int DM_60_VERSION =
    PORTMANTEAU(DM_60_MAJOR,DM_60_MINOR);

// The version in which pcurves in lofts where added to ACIS

const int LOFT_PCURVE_MAJOR = 6;
const int LOFT_PCURVE_MINOR = 0;
const int LOFT_PCURVE_VERSION =
    PORTMANTEAU( LOFT_PCURVE_MAJOR, LOFT_PCURVE_MINOR );

// The version in which an EELIST saves its owner and ownership policy

const int EELIST_OWNER_MAJOR = 6;
const int EELIST_OWNER_MINOR = 0;
const int EELIST_OWNER_VERSION =
    PORTMANTEAU( EELIST_OWNER_MAJOR, EELIST_OWNER_MINOR );

// The version in which an ANNOTATION saves whether members are hooked

const int ANNO_HOOKED_MAJOR = 7;
const int ANNO_HOOKED_MINOR = 0;
const int ANNO_HOOKED_VERSION =
    PORTMANTEAU( ANNO_HOOKED_MAJOR, ANNO_HOOKED_MINOR );

// The version in which patterns were added to ACIS

const int PATTERN_MAJOR = 7;
const int PATTERN_MINOR = 0;
const int PATTERN_VERSION =
    PORTMANTEAU( PATTERN_MAJOR, PATTERN_MINOR );

// The version in which tags were added to ENTITYs

const int ENTITY_TAGS_MAJOR = 7;
const int ENTITY_TAGS_MINOR = 0;
const int ENTITY_TAGS_VERSION =
    PORTMANTEAU( ENTITY_TAGS_MAJOR, ENTITY_TAGS_MINOR );
```

```
// The version at which strings were first distinguished
// in text files by a @ sign.

const int AT_MAJOR = 7;
const int AT_MINOR = 0;
const int AT_VERSION = PORTMANTEAU( AT_MAJOR, AT_MINOR );

// The version in which net law surface were added to ACIS

const int NET_LAW_MAJOR = 7;
const int NET_LAW_MINOR = 0;
const int NET_LAW_VERSION = PORTMANTEAU( NET_LAW_MAJOR, NET_LAW_MINOR );

// The version in which the ”bulletin” and ”bulletin_board” strings
// were removed from the history portion of the save file.

const int STRINGLESS_HISTORY_MAJOR = 7;
const int STRINGLESS_HISTORY_MINOR = 0;
const int STRINGLESS_HISTORY_VERSION =
    PORTMANTEAU( STRINGLESS_HISTORY_MAJOR, STRINGLESS_HISTORY_MINOR );

#endif
```