Chapter 40.
Options

Topic: Ignore

Options may be set to modify the behavior of ACIS. An option's value may be a flag (indicating an on/off state), a number (integer or real number), or a string. Options may be set in a Scheme application (such as Scheme AIDE) using the Scheme extension option:set; in the ACIS Test Harness using the command option; or in a C++ application using one of several API functions. Refer to the *3D ACIS Online Help User's Guide* for a description of the fields in the reference template.

address_debug

Option: Debugging, Modeler Control

Action: Sets the form of output addresses.

Name String: **address_debug**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: Selects whether or not to output actual entity addresses as well as list indices. Makes output position-independent, thus comparisons are easier.

Example:
```
; address_debug
; Turn off entity addresses
(option:set "address_debug" #f)
;; #t
```

annotations

Option: Modeler Control, Feature Naming

Action: Controls whether annotation entities are created.

Name String:	**annotations**		
Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: If this option is on (true), annotation entities are created during ACIS operations. If it is off, they are not. The annotations option has different default values for the release and the debug mode. It is on (true) by default in the debug mode. When the unhooking option is on (true), the annotation entities are not saved in the SAT file. However, unhooking does not occur until the outer most set of API_BEGIN and API_END macros. Hence the annotation entities would be saved in the SAT file, if the save is being performed within the same set of API_BEGIN and API_END macros as the annotated operations. This behavior could be overriden by explicitly turning off (false) the annotations option in debug mode.

Example:
```
; annotations
; Turn on annotation creation
(option:set ”annotations” #t)
;; #f
```

api_checking

Option: Modeler Control

Action: Controls whether or not arguments to APIs are checked.

Name String:	**api_checking**		
Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If this option is on, arguments to APIs are checked for validity. If it is off, they are not. This is a global option.

Example:
```
; api_checking
; Turn off API argument checking
(option:set ”api_checking” #f)
;; #t
```

backup_boxes

Option: Ignore

Action: Sets backup of mesh boxes.

Name String:	**backup_boxes**		
Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: Turning this off saves a lot of memory, but boxes must be recomputed as needed. Used in the tri3_msh_sur::copy_pointers function.

Example:
```
; backup_boxes
; Turn off mesh backup boxes
(option:set "backup_boxes" #f)
;; #t
```

bb_immediate_close

Option: Modeler Control, History and Roll

Action: Controls whether or not a bulletin board is closed off immediately when the call is made to the close_bulletin_board function.

Name String:	**bb_immediate_close**		
Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: Controls whether bulletin boards are closed in close_bulletin_board or in open_bulletin_board. Historically it has been done in the open, so that is the default. Switching the option on could result in finding errors related to missing API_BEGIN/API_ENDs a bit quicker, because this causes the bulletin board to be closed immediately in the outermost API_END instead of waiting for the next API_BEGIN.

Example:
```
; bb_immediate_close
; Close bulletin boards immediately
(option:set "bb_immediate_close" #t)
;; #f
```

binary_format

Option: Modeler Control, SAT Save and Restore

Action: Controls the format to use when writing ACIS part save files as binary.

Name String: **binary_format**

Scheme:	integer	0, 1, 2, 3, 4, 5, 6	0
Test Harness:	integer	0, 1, 2, 3, 4, 5, 6	0
C++:	int	0, 1, 2, 3, 4, 5, 6	0

Description: This option determines the format used when writing ACIS part save files in binary form (to .sab files). It controls whether the file is written in a 32 bit or 64 bit (word size) format and whether the file is written with big–endian or little–endian byte ordering. The word size only affects longs. Pointers are converted to (long) indices before writing. All other types are the same size on 32 and 64 bit platforms. This option does not affect the *reading* of binary files.

The possible values are:

0 Use the native format for the platform
1 Use big-endian byte ordering with native word sizes for the platform
2 Use little-endian byte ordering with native word sizes for the platform
3 Use big-endian byte ordering with 32 bit word sizes
4 Use little-endian byte ordering with 32 bit word sizes
5 Use big-endian byte ordering with 64 bit word sizes
6 Use little-endian byte ordering with 64 bit word sizes

Example:
```
; binary_format
; Using little–endian order and native word size
(option:set ”binary_format” 2)
;; 0
```

binary_read_format

Option: Modeler Control, SAT Save and Restore

Action: Controls the format to use when reading ACIS part save files as binary.

Name String: **binary_read_format**

Scheme:	integer	–1, 0, 1, 2, 3, 4, 5, 6	–1
Test Harness:	integer	–1, 0, 1, 2, 3, 4, 5, 6	–1
C++:	int	–1, 0, 1, 2, 3, 4, 5, 6	–1

Description: This option determines the format used when reading ACIS part save files in binary form (from .sab files). It controls whether the file is read in a 32 bit or 64 bit (word size) format and whether the file is read with big–endian or little–endian byte ordering. The word size only affects longs. Pointers are converted to (long) indices before being written to the .sab file. All other types are the same size on 32 and 64 bit platforms.

The possible values are:

–1 Determine the format automatically
0 Use the native format for the platform
1 Use big-endian byte ordering with native word sizes for the platform
2 Use little-endian byte ordering with native word sizes for the platform
3 Use big-endian byte ordering with 32 bit word sizes
4 Use little-endian byte ordering with 32 bit word sizes
5 Use big-endian byte ordering with 64 bit word sizes
6 Use little-endian byte ordering with 64 bit word sizes

Example:

```
; binary_read_format
; Using little–endian order and native word size
(option:set ”binary_read_format” 2)
;; –1
```

bl_envelope_surf

Option: Modeler Control, Blending

Action: Controls the type of blend surface used when a variable-radius blend is created.

Name String: **bl_envelope_surf**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If this option is on (true), a *rolling ball envelope* blend surface is created for the variable-radius blend. If it is off (false), a *rolling ball snapshot* blend surface is created.

Example:

```
; bl_envelope_surf
; Unite two blocks, then create two blends, one
; with bl_envelope_surf off, and one with it on
(define block1 (solid:block (position 0 5 0)
    (position 5 10 10)))
;; block1
(define block2 (solid:block (position 0 0 0)
    (position 10 10 5)))
;; block2
(solid:unite block1 block2)
;; #[entity 2 1]
(iso)
;; #[view 1077019000]
(zoom-all)
;; #[view 1077019000]
(ray:queue 35.8763 -51.7115 28.5678
    -0.408248 0.816497 -0.408248 1)
;; #[ray (35.8763 -51.7115 28.5678)
;; (-0.408248 0.816497 -0.408248)]
(define edge1 (pick-edge))
;; edge1
(solid:blend-edges edge1 2)
;; (#[entity 2 1])
(ray:queue 32.557 -50.0111 35.2878
    -0.408248 0.816497 -0.408248 1)
;; #[ray (32.557 -50.0111 35.2878)
;; (-0.408248 0.816497 -0.408248)]
(define edge2 (pick-edge))
;; edge2
(solid:blend-edges edge2 2)
;; (#[entity 2 1])
(define radii (abl:two-ends-rad 1 2))
;; radii


; Turn option off and create a blend
(option:set "bl_envelope_surf" #f)
;; #t
(ray:queue 29.4386 -53.3763 31.6758
    -0.408248 0.816497 -0.408248 1)
;; #[ray (29.4386 -53.3763 31.6758)
;; (-0.408248 0.816497 -0.408248)]
(define edge3 (pick-edge))
;; edge3
(define edges (blend:get-smooth-edges edge3))
;; edges
```

```
(blend:var-rad-on-edge edges 0 2)
;; (#[entity 6 1] #[entity 7 1] #[entity 8 1])
(blend:network edges)
;; #[entity 2 1]
; Turn option on and create another blend
(option:set "bl_envelope_surf" #t)
;; #f
(ray:queue 29.639 -52.2681 33.6918
    -0.408248 0.816497 -0.408248 1)
;; #[ray (29.639 -52.2681 33.6918)
;; (-0.408248 0.816497 -0.408248)]
(define edge4 (pick-edge))
;; edge4
(define edges (blend:get-smooth-edges edge4))
;; edges
(blend:var-rad-on-edge edges 0 2)
;; (#[entity 9 1] #[entity 10 1] #[entity 11 1])
(blend:network edges)
;; #[entity 2 1]
; Check convexity of edges
(ray:queue 34.0659 -49.9426 33.9158
    -0.408248 0.816497 -0.408248 1)
;; #[ray (34.0659 -49.9426 33.9158)
;; (-0.408248 0.816497 -0.408248)]
(define edge5 (pick-edge))
;; edge5
(edge:convexity edge5)
;; "convex"
(ray:queue 35.9327 -50.5213 30.8918
    -0.408248 0.816497 -0.408248 1)
;; #[ray (35.9327 -50.5213 30.8918)
;; (-0.408248 0.816497 -0.408248)]
(define edge6 (pick-edge))
;; edge6
(edge:convexity edge6)
;; "convex_smooth"
```

brief_comp_debug

Option: Ignore

Action: Sets how much information about a compcurv is printed.

Name String: **brief_comp_debug**

Scheme: boolean #f, #t #t

Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If on, a brief version of the information is printed by com_cur::debug.

Example:
```
; brief_comp_debug
; Turn off the brief debug option
(option:set "brief_comp_debug" #f)
;; #t
```

brief_curve_debug

Option: Modeler Control, Construction Geometry, Debugging

Action: Sets how much information about a curve is printed.

Name String: **brief_curve_debug**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If on, only a brief version of the curve information is printed by intcurve::debug.

Example:
```
; brief_curve_debug
; Turn off the brief debug option
(option:set "brief_curve_debug" #f)
;; #t
```

brief_mesh_debug

Option: Ignore

Action: Sets how much information about a mesh is printed.

Name String: **brief_mesh_debug**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If on, only a brief version of the mesh information is printed by msh_sur::debug_data.

Example:
```
; brief_mesh_debug
; Turn off the brief debug option
(option:set "brief_mesh_debug" #f)
;; #t
```

brief_pcurve_debug

Option: Modeler Control, Debugging, Construction Geometry

Action: Sets how much information about a pcurve is printed.

Name String: **brief_pcurve_debug**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If on, only a brief version of the pcurve information is printed by pcurve::debug.

Example:
```
; brief_pcurve_debug
; Turn off the brief debug option
(option:set "brief_pcurve_debug" #f)
;; #t
```

brief_surface_debug

Option: Modeler Control, Debugging, Construction Geometry

Action: Sets how much information about a spline surface is printed.

Name String: **brief_surface_debug**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If on, only a brief version of the spline surface information is printed by spline::debug.

Example:
```
; brief_surface_debug
; Turn off the brief debug option
(option:set ”brief_surface_debug” #f)
;; #t
```

careful

Option: Debugging, Modeler Control

Action: Controls whether or not extra geometry/topology checking is done.

Name String: **careful**

Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: If this option is on (TRUE), extra geometry/topology checking is done, at the expense of performance. This extra checking is not intended to be specific to any component or area of functionality—it may occur anywhere in ACIS. However, this option is currently being used only by sweeping to check for face–face intersections.

Example:
```
; careful
; Turn on checking
(option:set ”careful” #t)
;; #f
```

check_output

Option: Modeler Control, Spline Interface

Action: Determines the level of output generated by the body checker.

Name String: **check_output**

Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: Selects the level of output generated by the body checker. If this option is on, a more detailed output on the errors is generated and sent to stdout.

Example:

```
; check_output
; Turn on detailed output
(option:set "check_output" #t)
;; #f
```

compress_bb

Option: History and Roll, Modeler Control

Action: Controls bulletin board compression.

Name String: **compress_bb**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If on, bulletin boards are to be automatically compressed. This means that each bulletin board is automatically merged with any previous bulletin boards in the same delta state. This generally results in smaller memory usage. To maintain separate bulletin boards for each API, set this option to off during initialization.

Each set of operations within an outermost API_BEGIN/API_END block produces a bulletin board containing a bulletin for each entity created, changed, or deleted within that block. All operations on any given entity within the block go into a single bulletin. Thus, the bulletin indicates only the state of the entity before and after the entire block.

A call to API api_note_state creates a delta state containing all of the bulletin boards created since the previous call to api_note_state. If an entity has been modified in several different blocks, there will be several bulletins for that entity, each on a different bulletin board.

If the compress_bb option is on, at the end of each successful block the bulletins in the bulletin board created for that block are merged with those from the previous bulletin board, so they appear as though the operations occurred in the same block. This should save memory used by extra bulletins and backup copies of modified entities. It should also save time during roll back.

Example:

```
; compress_bb
; Turn off bulletin board compression
(option:set "compress_bb" #f)
;; #t
```

cone_param_range_v

Option: Modeler Control

Action: Controls whether to use a more accurate algorithm for computing cone surface parameter range.

Name String: **cone_param_range_v**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If this option is on, a more accurate—but slower—algorithm is used for computing the parameter range of a cone surface.

Example:
```
; cone_param_range_v
; Turn off cone_param_range_v
(option:set "cone_param_range_v" #f)
;; #t
```

convert_on_restore

Option: Modeler Control, SAT Save and Restore

Action: Controls whether to convert wires from the old to the new format during a restore operation.

Name String: **convert_on_restore**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: Refer to *Action*.

Example:
```
; convert_on_restore
; Do not convert
(option:set "convert_on_restore" #f)
;; #t
```

delete_forward_states

Option: Modeler Control, History and Roll

Action: Controls whether to delete all forward delta states.

Name String:	**delete_forward_states**		
Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If this option is on, all forward delta states are deleted when adding a new delta state to a history stream.

Example:
```
; delete_forward_states
; Turn off forward state deletion
(option:set "delete_forward_states" #f)
;; #t
```

error_no_input_tag

Option: Modeler Control, Feature Naming

Action: Controls whether or not an error occurs if inputs are not tagged (when annotations are activated).

Name String:	**error_no_input_tag**		
Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: This option affects the behavior when annotations are activated (e.g., using option annotations); it has no affect when annotations are not turned on (because annotations are not created). If this option is on (true), inputs must by tagged (by adding an ATTRIB_TAG or an entity derived from it) by the caller before calling the ACIS operation. If not tagged, a sys_error occurs when creating an annotation that references the input. This can be a useful debugging tool. If this option is off (false), tags are generated as needed.

Example:
```
; error_no_input_tag
; Turn on error if not tagged
(option:set "error_no_input_tag" #t)
;; #f
```

fitol_curve_interp

Option: Modeler Control, Spline Interface

Action: Sets the fit tolerance for spline curve interpolation.

Name String: **fitol_curve_interp**

Scheme:	real	See *Description*	–1.0
Test Harness:	double	See *Description*	–1.0
C++:	double	See *Description*	–1.0

Description: The fit tolerance is set based on the value of this option, as follows:

Option Value	Fit Tolerance Used
Greater than 0.0	Option value
Equal to 0.0	SPAresfit * arc length; where arc length is the length of the path created by connecting the initial fit points with line segments
Less than 0.0	SPAresfit

If the resulting fit tolerance is less than 10*SPAresabs, then 10*SPAresabs is used.

Example:

```
; fitol_curve_interp
; Set fit tolerance to use arc length * SPAresfit
(option:set "fitol_curve_interp" 0.0)
;; –1
```

fix_pcurves

Option: Modeler Control, Intersectors

Action: Controls whether or not pcurves are corrected when validity checks are performed.

Name String: **fix_pcurves**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: This option is used whenever validity checks are performed on a pcurve. This happens whenever a CCS (curve–curve intersection on a surface) is done. The pcurve checking code checks various properties of the pcurve. If this option is on (true), the checking code will attempt to correct any properties that are not satisfied. If this option is off (false), there is no attempt to correct the pcurve.

Example:
```
; fix_pcurves
; Turn off pcurve fixing
(option:set ”fix_pcurves” #f)
;; #t
```

history_checks

Option: Modeler Control, History and Roll

Action: Controls how history checks are reported.

Name String: **history_checks**

Scheme:	integer	See Description	0
Test Harness:	integer	See Description	0
C++:	int	See Description	hs_checks_off

Description: This option controls the reporting of history checks (including checking for mixed history streams). The valid values are defined in the enumerated type hs_checks_level:

0 = hs_checks_off, no checking is done
1 = hs_checks_warning, any identified problems result in a warning
2 = hs_checks_error, any identified problems result in an error being thrown

Example:
```
; history_checks
; Change history check reporting
(option:set ”history_checks” 2)
;; 0
```

intcurve_save_approx_level

Option: Modeler Control, SAT Save and Restore

Action: Controls the level of information stored in the SAT file for intcurves.

Name String: **intcurve_save_approx_level**

Scheme:	string	See *Description*	“optimal”
Test Harness:	string	See *Description*	“optimal”
C++:	char*	See *Description*	“optimal”

Description: This option controls the amount of data stored in the SAT file for intcurves. In particular, it controls whether the approximating geometry for a spline curve is stored in full, in summary form, or not at all. If the approximating geometry is stored in full, then the SAT file will be large, but regenerating the part from the SAT file will be relatively fast. If the approximating geometry is not stored at all, then the SAT file will be at its minimum size, but parts may take a long time to regenerate because the approximating geometry must be completely recalculated. The summary form is a compromise. The SAT files will be only slightly larger than when no approximating geometry is stored, and regeneration is nearly as fast as when the full geometry is stored.

In this discussion, *regenerate* means to restore the data and prepare it for use. In release 5.0, approximating geometry may not be recalculated during the restore, but it will be recalculated when it is first required. Therefore, if approximating geometry is stored in full, the actual restore will be fast, but the part may not be "ready for use" until the approximating geometry has been recalculated.

The argument to this option is a string. Possible values are:

"full"	Save the complete approximating geometry.
"summary"	Save a summary form of the approximating geometry.
"none"	Do not save the approximating geometry.
"historical"	Preserve the historical behavior; i.e., save the approximating geometry if and only if this was done in pre-5.0 versions.
"optimal"	Allow ACIS to decide the level at which approximating geometry is saved.

ACIS may override the setting of this option for a particular geometry type. Typically, this will be because the geometry type requires the approximating geometry as a fundamental part of its definition and cannot exist without it. For example, if this option is set to "none," exact intcurves (exact_int_cur) will still be saved in full, because they would otherwise be undefined.

The possible values for this option are defined in the enumeration save_approx_level and in the corresponding enum_entry structure save_approx_entries (and its enum_table save_approx_map), which defines the strings and maps them to the enumeration. Refer to the description of the Enumeration Template in the *3D ACIS Online Help User's Guide* for more information about the enum_entry structure.

save_approx_full . . . . . "full"
save_approx_summary . . . . . "summary"
save_approx_none . . . . . "none"
save_approx_historical . . . . . "historical"
save_approx_optimal . . . . . "optimal"

Example:

```
; intcurve_save_approx_level
; Set to full save
(option:set "intcurve_save_approx_level" "full")
;; "optimal"
```

logging

Option: Modeler Control, History and Roll

Action: Sets whether bulletin boards and delta states are to be visible at the application level.

Name String: **logging**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If off, each bulletin board is deleted as soon as the next is opened, and application functions to get bulletin boards or delta states always return NULL.

Example:

```
; logging
; Turn off logging
(option:set "logging" #f)
;; #t
```

new_dangling_wires

Option: Modeler Control, SAT Save and Restore

Action: Converts dangling wires to either new or old style.

Name String: **new_dangling_wires**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: Used when restoring old-style save files. If off, converts backwards; if on, converts forwards.

Example:
```
; new_dangling_wires
; Convert backwards
(option:set ”new_dangling_wires” #f)
;; #t
```

new_transform_method

Option: Modeler Control, Model Geometry, Transforms

Action: Sets the method used for transformations.

Name String: **new_transform_method**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If on, uses the new streamlined method for performing transformations. If off, uses the old method.

Example:
```
; new_transform_method
; Use old method
(option:set ”new_transform_method” #f)
;; #t
```

print_entity_type

Option: Modeler Control, Scheme AIDE Application, Debugging

Action: Controls how an entity is printed in Scheme output.

Name String: **print_entity_type**

Scheme:	boolean	#f, #t	#f

Test Harness: Not applicable

C++: Not applicable

Description: If this option is on (true), the type name of the entity (e.g., face, body, etc.) gets printed in Scheme output instead of just the string “entity”. If it is off, Scheme simply prints “entity”.

Example:

```
; print_entity_type
; Create a body and print some entity information
(define b (solid:block (position 0 0 0)
    (position 1 1 1)))
;; b
b
;; #[entity 2 1]
(entity:faces b)
;; (#[entity 3 1] #[entity 4 1] #[entity 5 1]
;; #[entity 6 1] #[entity 7 1] #[entity 8 1])
;
; Turn the option on and repeat
(option:set ”print_entity_type” #t)
;; #f
(define b (solid:block (position 0 0 0)
    (position 1 1 1)))
;; b
b
;; #[body 9 1]
(entity:faces b)
;; (#[face 10 1] #[face 11 1] #[face 12 1]
;; #[face 13 1] #[face 14 1] #[face 15 1])
```

regen_skin_approx

Option: Modeler Control, SAT Save and Restore

Action: Controls whether to regenerate the approximating surface during retrieval of a skin surface.

Name String: **regen_skin_approx**

Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: If this option is on, the approximating surface of a skin_spl_sur is regenerated. Surfaces created with older versions of ACIS may have bad approximating surfaces.

Example:

```
; regen_skin_approx
; Regenerate the surface
(option:set ”regen_skin_approx” #t)
;; #f
```

remesh

Option: Ignore

Action: Sets automatic remeshing during a Boolean.

Name String: **remesh**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: Any compcurv intersections are used to break up the mesh faces on which they lie. Breaks up the triangles along a boundary curve to be compatible with the segments of that curve. Limited to straight segments. When on, this will happen automatically during a Boolean. When off, the user must remesh the body.

Example:

```
; remesh
; Turn off automatic remeshing
(option:set ”remesh” #f)
;; #t
```

restore_locale

Option: Modeler Control, SAT Save and Restore

Action: Sets the localization properties (language locale) to use when restoring save files (.sat).

Name String: **restore_locale**

Scheme:	string	See Description	“C”
Test Harness:	string	See Description	“C”
C++:	char*	See Description	“C”

Description: This provides the ability to restore .sat files that were saved using a locale other than the “C” locale. It results in a call to the C standard library function setlocale *if a restore operation is performed*. The locale is reset to its original value following the restore.

The argument to this option is a string. Possible values are:

"C"	Uses the standard environment for C.
"" (empty string)	Uses the system's native environment.
"<*other valid locale name*>"	Is a string specifying some other locale. Valid values are *system dependent* locale names.

For example, in HP-UX version 10.01, a locale name conforms to ISO standards and identifies the language (with a 2-character code), country or territory (with a 2-character code), and the codeset of that locale. Examples of valid locale names for an HP-UX (10.01) system include "ar_DZ.arabic8" (for the Arabic language in Algeria, using the arabic8 codeset) "iw_IL.hebrew8" (Hebrew, Israel, hebrew8), and "de_DE.iso88591" (German, Germany, ISO 8859/1).

Example:

```
; restore_locale
; On HP-UX 10.01, set locale to German, ISO codeset
(option:set "restore_locale" "de_DE.iso88591")
;; "C"
```

res_near_tangent

Option: Modeler Control, Tolerant Modeling

Action: Sets the tolerant modeling resolution (tolerance) for determining if an edge is considered tangent.

Name String: **res_near_tangent**

Scheme:	real	01234567890.,<>=	0.0175
Test Harness:	double	01234567890.,<>=	0.0175
C++:	double	01234567890.,<>=	0.0175

Description: This option is used for tolerant modeling, and is a *system-wide* setting for the tolerance to use in testing whether or not an edge is considered tangent (based on the angle between normals). Tolerant modeling algorithms throughout the system may use this value. This option may be used in conjunction with other options that control specific tolerant modeling functionality. For example, this option may have no effect on certain operations if tolerant modeling is not specifically enabled for that functionality (e.g., blending, skinning, local operations, etc.).

The value given is specified in radians, with a logical range of 0.0 to 2pi. In practice, this will be a small number close to zero. The default setting of 0.0175 is approximately 1 degree.

Example:

```
; res_near_tangent
; Increase the tolerance to 2 degrees
(option:set ”res_near_tangent” .035)
;; 0.0175
```

ret_directory

Option: Modeler Control, SAT Save and Restore

Action: Sets the default directory for retrieve files in the ACIS Test Harness.

Name String: **ret_directory**

Scheme: Not applicable

Test Harness: string valid pathname ””

C++: Not applicable

Description: This option only applies to the ACIS Test Harness. This is the default directory name for files retrieved using the retrieve command.

Example: Not applicable

save_box

Option: Modeler Control, SAT Save and Restore

Action: Sets writing the bounding boxes to the save file.

Name String: **save_box**

Scheme: boolean #f, #t #t

Test Harness: integer 0, 1 1

C++: Not applicable

Description: When on, saves the bounding boxes in the SAT file. To improve performance when performing operations on restored models, bounding boxes(includes uv bounding boxes) are saved in the SAT file by default. However this can be avoided by setting this option to off.

Example:
```
; save_box
; Do not write the bounding boxes to the file
(option:set ”save_box” #f)
;; #t
```

save_directory

Option: Modeler Control, SAT Save and Restore

Action: Sets the default directory for save files in the ACIS Test Harness.

Name String: **save_directory**

Scheme: Not applicable

Test Harness:	string	valid pathname	“”

C++: Not applicable

Description: This option only applies to the ACIS Test Harness. This is the default directory name for files created using the save command.

Example: Not applicable

save_entity_count

Option: Modeler Control, SAT Save and Restore

Action: Sets writing the entity count to the save file.

Name String: **save_entity_count**

Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: When on, forces the number of entities saved to be written to the file header. Thus, when on, the save procedure must reposition the file, so you may not be able to save to some targets.

Example:
```
; save_entity_count
; Write the entity count to the file
(option:set ”save_entity_count” #t)
;; #f
```

save_old_sab

Option: Modeler Control, SAT Save and Restore

Action: Sets format for binary save files.

Name String: **save_old_sab**

Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: When on, saves binary files in the old format which does not support unknown entity types. Only truly useful in testing.

Example:
```
; save_old_sab
; Use old binary format
(option:set "save_old_sab" #t)
;; #f
```

save_unknown_subtype_as_approx

Option: Modeler Control, SAT Save and Restore

Action: Sets how unknown subtypes are saved in old save file formats.

Name String: **save_unknown_subtype_as_approx**

Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: When on, saves unknown spl_sur or int_cur to old versions as an exact, based on the approximate information.

Example:
```
; save_unknown_subtype_as_approx
; Save unknowns as approximates
(option:set "save_unknown_subtype_as_approx" #t)
;; #f
```

save_version

Option: Modeler Control, SAT Save and Restore

Action: Sets the ACIS version to use for writing save files in the ACIS Test Harness.

Name String: **save_version**

Scheme: Not applicable

Test Harness: integer 3-digit ACIS version installed version

C++: Not applicable

Description: This option only applies to the ACIS Test Harness. The version controls the save file format. The first digit is the major version number. The second and third digits are the minor version number. For example, version 2.1 is represented as 201, and version 3.0 is represented as 300. The default value is the currently installed version of ACIS. In C++ applications, the API api_save_version is used to set the version.

Example: Not applicable

sequence_save_files

Option: Modeler Control, SAT Save and Restore

Action: Sets sequence numbers in save files.

Name String: **sequence_save_files**

Scheme: boolean #f, #t #f

Test Harness: integer 0, 1 0

C++: logical FALSE, TRUE FALSE

Description: When on, enables the writing of sequence numbers in .sat files

Example:
```
; sequence_save_files
; Write sequence numbers to save file
(option:set "sequence_save_files" #t)
;; #f
```

spline_save_approx_level

Option: Modeler Control, SAT Save and Restore

Action: Controls the level of information stored in the SAT file for spline surfaces.

Name String: **spline_save_approx_level**

Scheme: string See *Description* "optimal"

Test Harness:	string	See *Description*	“optimal”
C++:	char*	See *Description*	“optimal”

Description: This option controls the amount of data stored in the SAT file for spline surfaces. In particular, it controls whether the approximating geometry for a spline surface is stored in full, in summary form, or not at all. If the approximating geometry is stored in full, then the SAT file will be large, but regenerating the part from the SAT file will be relatively fast. If the approximating geometry is not stored at all, then the SAT file will be at its minimum size, but parts may take a long time to regenerate because the approximating geometry must be completely recalculated. The summary form is a compromise. The SAT files will be only slightly larger than when no approximating geometry is stored, and regeneration is nearly as fast as when the full geometry is stored.

In this discussion, *regenerate* means to restore the data and prepare it for use. In release 5.0, approximating geometry may not be recalculated during the restore, but it will be recalculated when it is first required. Therefore, if approximating geometry is stored in full, the actual restore will be fast, but the part may not be “ready for use” until the approximating geometry has been recalculated.

The argument to this option is a string. Possible values are:

“full”	Save the complete approximating geometry.
“summary”	Save a summary form of the approximating geometry.
“none”	Do not save the approximating geometry.
“historical”	Preserve the historical behavior; i.e., save the approximating geometry if and only if this was done in pre-5.0 versions.
“optimal”	Allow ACIS to decide the level at which approximating geometry is saved.

ACIS may override the setting of this option for a particular geometry type. Typically, this will be because the geometry type requires the approximating geometry as a fundamental part of its definition and cannot exist without it.

The possible values for this option are defined in the enumeration save_approx_level and in the corresponding enum_entry structure save_approx_entries (and its enum_table save_approx_map), which defines the strings and maps them to the enumeration. Refer to the description of the Enumeration Template in the *3D ACIS Online Help User's Guide* for more information about the enum_entry structure.

save_approx_full . . . . . "full"
save_approx_summary . . . . . "summary"
save_approx_none . . . . . "none"
save_approx_historical . . . . . "historical"
save_approx_optimal . . . . . "optimal"

Example:
```
; spline_save_approx_level
; Set to full save
(option:set "spline_save_approx_level" "full")
;; "optimal"
```

split_curves

Option: Modeler Control, Construction Geometry

Action: Sets curve splitting in the curve::split function.

Name String: **split_curves**

Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: When on, enables curve splitting in the curve::split function. If off, curve::split does nothing and returns NULL.

Example:
```
; split_curves
; Turn on curve splitting
(option:set "split_curves" #t)
;; #f
```

string_check

Option: Modeler Control

Action: Sets how NULL strings are handled.

Name String:	**`string_check`**		
Scheme:	Not applicable		
Test Harness:	Not applicable		
C++:	logical	FALSE, TRUE	FALSE
Description:	Selects “fixup” or “collapse” option. The general fixup treats NULL strings as empty ones, making a check/fix function useful.		
Example:	Not applicable		

sweep_selfint

Option: Modeler Control, Sweeping

Action:	Sets self intersection checks while evaluating the sweep surface.		
Name String:	**`sweep_selfint`**		
Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE
Description:	When on, enables some limited self intersection checks while evaluating the sweep surface. This is normally turned off and is turned on only during the surface construction.		

Example:

```
; sweep_selfint
; Turn on self intersection checks
(option:set ”sweep_selfint” #t)
;; #f
```

test_share

Option: Modeler Control, SAT Save and Restore

Action:	Sets detecting and sharing of identical objects when restoring save files.		
Name String:	**`test_share`**		
Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1

C++:	logical	FALSE, TRUE	TRUE

Description: When on, int_cur and spl_sur types are compared with those that have already been restored to determine if they are identical to a previously restored int_cur or spl_sur. when restoring save files. This option significantly reduces the size of retrieved bodies and aids subsequent operations, but it can be expensive and can become noticeable when restoring large parts. Turning it off will speed up the restore process. However, the amount of memory required to restore a model will be larger, since sharing of geometry is not taking place. Also, evaluations of geometry during modeling operations may take longer because the test for coincidence will take place each time the objects are evaluated. This may cause the test to happen many times instead of once when the model is loaded. On is the default.

Example:
```
; test_share
; Turn off share testing
(option:set "test_share" #f)
;; #t
```

tight_sphere_box

Option: Modeler Control, Model Geometry

Action: Sets calculation of a tight bounding box for a sphere.

Name String: **tight_sphere_box**

Scheme:	boolean	#f, #t	#f
Test Harness:	integer	0, 1	0
C++:	logical	FALSE, TRUE	FALSE

Description: When on, enables calculation of a tight bounding box for a sphere.

Example:
```
; tight_sphere_box
; Turn on sphere bounding box
(option:set "tight_sphere_box" #t)
;; #f
```

tight_torus_box

Option: Modeler Control, Model Geometry

Action: Sets calculation of a tight bounding box for a torus.

Name String: **tight_torus_box**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: When on, enables calculation of a tight bounding box for a torus.

Example:
```
; tight_torus_box
; Turn off torus bounding box
(option:set "tight_torus_box" #f)
;; #t
```

torus_param_range

Option: Modeler Control, Construction Geometry

Action: Sets whether or not a box is used to find the torus parameter range.

Name String: **torus_param_range**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: When on, the box supplied to torus::param_range is used to find a reasonably small parameter range for the torus (otherwise, the entire torus range is returned).

Example:
```
; torus_param_range
; Turn option off
(option:set "torus_param_range" #f)
;; #t
```

unhook_annotations

Option: Modeler Control, Feature Naming

Action: Controls whether annotations are automatically unhooked from their entities.

Name String: **unhook_annotations**

Scheme:	boolean	#f, #t	#t
Test Harness:	integer	0, 1	1
C++:	logical	FALSE, TRUE	TRUE

Description: If this option is on (true), annotations are automatically unhooked from their entities at the outermost API_END enclosing an operation. This is done by losing all ANNOTATION_ATTRIBs, as in API function api_unhook_annotations.

Example:

```
; unhook_annotations
; Turn off automatic unhook
(option:set ”unhook_annotations” #f)
;; #t
```