Parasolid V34.1

KI Programming Reference Manual

January 2022

Important Note

Francis House
112 Hills Road
Cambridge CB2 1PH
United Kingdom
Tel: +44 (0)1223 371555
email:
parasolid.support.plm@siemens.com
Web: www.parasolid.com

Trademarks

Table of Contents

Parasolid KI Programming Concepts

1

1.1 Introduction

This chapter explains the format of the specifications and the conventions that are used by the Parasolid Kernel Interface. This is the original interface to Parasolid, and its functionality is now almost completely replaced by the PK (Parasolid Kernel) Interface. However, the information contained in this manual will be useful to users maintaining existing applications which call KI routines, and those requiring functionality which has not yet been replaced by PK functions.

Additional Reading

For further information see the 'Parasolid Concepts' chapter of the Parasolid Functional Description manual. This information is fundamental to the understanding of Parasolid programming concepts and should therefore be read and understood.

1.2 Compatibility with Fortran

Although PARASOLID is written in 'C', the Kernel Interface has been designed so that it can be called either from 'C' or from suitable implementations of Fortran. To facilitate this compatibility with Fortran, the kernel interface follows these conventions:

- Kernel Interface routine names consist of six characters, being upper case letters or digits.
- Kernel Interface routines do not return a function value.
- Kernel Interface routines require only integer, double and character data types as arguments (the 'underlying' data types) and the only data structure used is the array (note that array subscripts in 'C' start at zero).
- all arguments are passed by reference.
- where a Kernel Interface routine refers to a constant value, it uses a mnemonic token. Token values form part of the documented interface.

> **Note:** PARASOLID documentation is written in terms of 'C' conventions - for example, arrays are assumed to start at index 0.

1.3 Format of routine headers

Each function header consists of:

- routine name
- brief description
- list of received arguments
- list of returned arguments
- list of specific errors
- detailed description

These components of the header are described in the following sub-sections.

1.3.1 Routine name

Each routine in the Kernel Interface has a six character name, consisting of upper case letters and digits.

1.3.2 Brief description

This provides a summary of what each Kernel Interface routine does.

1.3.3 List of received arguments

Received arguments pass information from the application program to the Kernel.

The application program must declare a variable of the appropriate type for each argument, and set this to the required value. The address of the variable is then passed to the kernel function for each argument; in ‘C’, this can be done by use of the address-of operator “&” for scalars, and passing the array pointer itself for an array.

Received arguments are not modified by the Kernel.

1.3.4 List of returned arguments

Returned arguments pass information from the Kernel to the application program.

The application program must declare variables to correspond to each of the returned arguments in a Kernel routine. The address of a returned argument is passed to the kernel function in the same way as for a received argument.

The contents of returned arguments will always be set after a function call even if an error has been detected by the Kernel routine. For a non-zero ifail they will be set to the default values.

The last returned argument for each Kernel routine is an error code (‘ifail’). This error code will be returned as zero if the call was successful, but will have a non-zero value if some error was detected. See later section on error returns for further details.

Argument format

Both received and returned arguments are declared in the form: typedef name followed by, variable name followed by, comment

- typedef name - The typedef is either one of the 'underlying' types: int, double, char or is a name of the form KI_int_... KI_dbl_... KI_chr_... KI_cod_... KI_vec_... KI_tag_... (where ... is an abbreviation for "one or more characters"). More details are given in the section dealing with special typedefs.

The typedef name may be enclosed by or may contain angle brackets < >. Their significance is explained in the section 'meaning of Angle Brackets'.

- variable name - The variable name is either preceded by an asterisk * or is followed by an expression enclosed by square brackets [ ].

The first form identifies a scalar variable. The asterisk indicates that the function requires the address of an appropriate variable as its argument.

The second form identifies a variable which is an array. The calling program must supply an array which is large enough to hold the received or returned argument. The array dimension is normally given by the expression within the square brackets; in other cases, the array dimension is given in the detailed description of the function. Note that arrays are always one dimensional and that all documentation uses the 'C' convention that array subscripts start from zero.

- comment - The comment describes the use of an argument, it can extend over more than one line.

When a routine requires options (iopts) or properties (props) and option data (optdta) or property data (pdata) to be entered, it is imperative that both argument arrays are of equal length. For example, the routine for creating a B-curve by splining (CRSPPC) requires properties and their data, therefore if the properties of the curve were declared as:

```
props [0] = PAPRCV;
props [1] = PAPRKT;
```

the data associated with these two properties would be declared as:

```
pdata [0] = NULTAG;
pdata [1] = knot_list;
```

where

- NULTAG indicates that there isn't any data to be supplied with the property token PAPRCU, which is used to force the curve to be cubic
- knot_list is a KI list containing two doubles specifying the knot vector of the curve

1.3.5 List of specific errors

The list of specific errors gives a list of possible error code returns for that particular Kernel Interface routine. Note that other error code returns are also possible; for example if an argument does not conform to the specified typedef. See the later section on error returns for details.

1.3.6 Detailed description

This explains what the Kernel routine does in more detail. Where reference is made to particular routine arguments, their names are delimited by single quotation marks ‘ ’. If a routine makes reference to other Kernel Interface routine names or to token values, these use upper case letters or digits and are six characters long.

1.4 Special typedefs

As mentioned previously, arguments to kernel interface functions are commonly defined in terms of special typedefs. The purpose of such special typedefs is to specify more clearly what the argument is and what values it may have.

If a received argument is declared in the routine header in terms of a particular typedef, the Kernel can apply run-time argument checking if this option has been selected by the SEINTP routine, option SLIPCH. In the case of an array argument, such typedef checks will be applied to each element of the array.

A summary of the different typedefs is listed in the appendix C. Each entry includes the error codes which can be returned by the Kernel if an argument fails the check for that typedef.

1.4.1 Classes of special typedefs

The special typedefs can be sub-divided into a number of different classes depending on their prefix (KI_int, KI_dbl, KI_chr, KI_cod, KI_vec, KI_tag, KI_tag_list). These classes are described in the succeeding sub-sections.

- KI_int_...

  The special typedefs which are in the form KI_int_... denote that the application program must supply a variable which is an integer and that the initial value of a received variable must be in a particular range.

  e.g. If a received variable is declared as a pointer of typedef KI_int_order, this denotes that an integer must be supplied which represents the order of a B-curve or B-surface (which must be >= 2).

  The errors listed in the appendix A show that a routine which uses this typedef can return a ‘typedef error’ of ‘KI_order_lt_2’ in addition to any of the specific errors listed in the header.

- KI_dbl_...

  The special typedefs which are in the form KI_dbl_... denote that the application program must supply a variable which is a double and that the initial value of a received variable must be in a particular range.

  e.g. If a received variable is declared as a pointer of typedef KI_dbl_sc_fact, this denotes that a double must be supplied which represents the scaling factor for a transformation (which must be > 0.0).

The errors listed in the appendix A show that a routine which uses this typedef can return a 'typedef error' of 'KI_sc_factor_le_0' in addition to any of the specific errors listed in the header

- KI_chr_...

  The special typedefs which are in the form KI_chr_... denote that the application program must supply a variable which is a char and that the initial value of a received variable must be in a particular range.

  e.g. If a received variable is declared as a pointer of typedef KI_chr_filename, this denotes that a single character or all characters in an array must be suitable for inclusion in a filename.

- KI_cod_...

  The special typedefs which are in the form KI_cod_... denote that the application program must supply a variable which is an integer and that the initial value of a received variable must be set to one of a limited number of code or ‘token’ values.

  e.g. If a received variable is declared as a pointer of typedef KI_cod_tyge this denotes that the variable of that type must be set to one of the token values represented by mnemonics TYGEPT, TYGECU, TYGESU, TYGETF.

  The token mnemonics and corresponding values denoted by each of the KI_cod_.... typedefs are listed in appendix C.

Note: Logical flags are represented by variables of typedef KI_cod_logical. This form of typedef implies that a variable must be supplied which is of type integer and must be set to one of the token values KI_false or KI_true, whose values are given in Appendix B.

- KI_vec_...

  The special typedefs which are in the form KI_vec_... denote that the application program must supply an array of type double of which the dimensioned length is a multiple of 3.

Note: The Kernel does not require directional vectors to be normalized although it may require certain types of vectors to be non zero.

e.g. If a received variable is declared as an array of pointers of typedef KI_vec_normal which is of length 2, this denotes that the application program must supply an array of doubles of length 6, with elements 0,1,2 representing the first vector and elements 3,4,5 representing the second. Neither element must be zero (or the typedef error code ‘KI_null_direction’ will be returned).

- KI_tag_... (excluding lists)

  The special typedefs which are in the form KI_tag_... denote that the application program must supply an integer variable to hold a tag. If the argument is a received variable, it must contain a valid tag and the tag must be of the appropriate type.

  Tag typedefs are either specific (such as KI_tag_body) or general (such as KI_tag_geometry).

The most general class of tag typedef is KI_tag. This allows all forms of tag (including deleted tags) and does not involve any validity checks.

- KI_tag_list_...

  If the typedef names are in the form KI_tag_list..., this implies that the argument is the tag of a list of integers, doubles, vectors, characters or is the tag of a list of tags. The contents of the list are implied by the last part of the typedef name (which replaces the ... shown above). This part of the name is called the underlying type of the list, e.g.:

  KI_tag_list_int implies a tag which refers to a list of integers

  KI_tag_list_face implies a tag which refers to a list of faces

  KI_tag_list_geometry implies a tag which refers to a list of geometric entities (curves, surfaces, points, transformations)

  - list entries

    The Kernel will apply the same checks to each entry in the list as it would if the variable had been declared by the underlying typedef.

    If the routine returns an error code and the error has occurred as a result of one of the entries in a list being in error, the entry number of this item can be returned by the `output last error' function OULERR. Note that the entry numbers start from one (unlike array subscripts).

    In most cases, Kernel routines do not allow empty lists to be passed as received arguments. Exceptions are the list-handling routines PTINLI, PTRLLI, PTTGLI, SRCHIL, SRCHRL, and SRCHTG.

    If a routine returns a list, the application program must supply an integer variable into which the Kernel will write the tag of the new list.

  - lists of one tag

    Where received arguments are specified as lists of tags, the Kernel will accept the tag of an entity (which is acceptable to the underlying typedef) as an alternative to passing the tag of a list which contains that entity.

    e.g. if a received argument is described by typedef KI_tag_list_body, this implies that it will also accept single tags of typedef KI_tag_body.

    This is for the convenience of the application programmer as it avoids the need to create and to delete lists of tags containing single items.

> **Note:** This does not apply when the received arguments are specified as a list of lists where it is not valid to shed a level of the identifying direction

1.4.2 Meaning of angle brackets

If a typedef name is enclosed by angle brackets, this denotes that the received argument is allowed to contain a particular ‘default’ value associated with the typedef (where this is applicable). The interpretation of the default value depends upon the category of typedef and upon whether a particular typedef makes use of it. Typedefs of the form KI_chr_..., KI_cod_..., and the ‘underlying’ types (int, double and char) do not use the default value mechanism, so are never enclosed by angle brackets. The sub-sections which follow describe the interpretation of the angle brackets for the classes of typedefs for which they are used.

- numeric defaults -

  If the typedef is of the form KI_int_..., KI_dbl_... or KI_vec_..., the default value is integer zero, floating point zero or the zero vector.

  If a routine argument is declared with the typedef enclosed by angle brackets, this specifies that the value of the associated variable can be zero.

  If no angle brackets are given, the question as to whether or not zero values are allowed, depends upon the way in which the range of values allowed by the typedef have been defined, e.g. if an argument is declared as:

  KI_dbl_distance *height, the height must be > 0.0 but for

  <KI_dbl_distance> *height, the height must be >= 0.0

  The question of whether or not a typedef makes a special case of default values can usually be resolved by examining the error codes associated with a particular typedef, which are listed in appendix A.

  If a typedef does not make a special case of the default value, the declaration will not contain angle brackets:

  e.g. if an argument is declared as:

  KI_vec_position *origin, the zero vector is not a special case, but for

  KI_vec_normal *direct, the zero vector is specifically excluded

- null tags (excluding lists)

  If the typedef is of the form KI_tag_..., the default value is the null entity token NULTAG which is mnemonic, and its value is defined in Appendix B, e.g.:

  KI_tag_body *entity, the variable must contain the tag of a body

  <KI_tag_body> *entity, the variable can contain the tag of a body or can be set to the null tag (NULTAG)

- null tags in lists

  If the typedef is of the form KI_tag_list... and the underlying type of the list is of one of the types of tags, the function headers can specify that the list can contain null tags by enclosing the name of the underlying type with further angle brackets. Examples:

  KI_tag_list_<body> *entities, specifies a list which can contain body tags or null tags but which cannot be the null tag itself (i.e. the list must be valid)

  <KI_tag_list_<body>> *entities specifies a list which can contain bodies or null tags, or which can itself be the null tag (i.e. the list need not exist)

> **Note:** Both examples will also allow a single body tag to be given instead of a list of bodies by the concession that allows one tag of the underlying type to be given instead of a list

1.5 Error returns

Each Kernel routines returns an error code in its list of returned arguments. By convention, this variable is named ‘ifail’ and is the last argument of the routine. Error codes are defined as mnemonic codes (listed in appendix B); where possible, application programs should use these error code mnemonics instead of the associated numeric values.

The Kernel routine OULERR can be called to get further information (e.g. the name of the erroneous argument) for the most recent kernel error; this may help the application programmer to identify the problem.

The error codes fall into five categories:

- success
- type validation errors
- exception conditions
- implicit receive errors
- specific errors

The first category implies that the operation was successful.

The other four categories imply that the operation was not successful and that the routine has taken no further action (other than setting the values of returned arguments to default values). These four categories are described in the sub-sections which follow.

1.5.1 Type validation errors

Type validation error codes are not listed explicitly in the routine header but can be derived by cross referencing the typedef names in the header with the error codes which are associated with each typedef name. The list of typedef names appears in appendix C.

An example of this type of error is where the header describes an argument as (a pointer to) a body tag. If the Kernel routine detects that it has been passed an inappropriate tag, it will return an error code which belongs to the type validation class of errors. In general, these codes are not listed in the list of specific errors which apply to a function.

1.5.2 Exception conditions

The codes for exception conditions are not listed explicitly in the function header since they apply to almost every type of function and can occur at any time. An example of this type of error is if the operating system is unable to supply sufficient virtual memory.

The exception error condition codes are:

- KI_system_error
- KI_memory_full
- KI_null_arg_address
- KI_modeler_not_started
- KI_modeler_not_stopped
- KI_recursive_call
- KI_aborted
- KI_run_time_error
- KI_fatal_error

Explanation of each code is given in Appendix B, “Kernel Interface Error Codes”.

1.5.3 Implicit receive errors

When an assembly is read using GETMOD, the sub-parts of the assembly are not received immediately. However, if some subsequent Kernel routine requires one of the sub-parts to be in memory, this sub-part will be received implicitly as a side-effect of the operation.

For example, suppose we receive an assembly, and then call the MASSPR function to determine its center of gravity. In order to calculate the center of gravity, the MASSPR function must implicitly receive all the sub-parts of the assembly. Now suppose further that there is a problem in receiving one of the sub-parts; this will result in an “implicit-receive” error being returned by MASSPR. A similar situation can occur with many other Kernel routines.

Because they can occur in many functions, the implicit-receive error codes are not documented in the routine headers. The possible error codes are:

- KI_wrong_format
- KI_usfd_mismatch
- KI_bad_key
- KI_key_not_found
- KI_wrong_version
- KI_cyclic_assy
- KI_keyed_part_mismatch
- KI_size_mismatch
- KI_attr_defn_mismatch
- KI_corrupt_file
- KI_receive_failed

See GETMOD or Appendix B, “Kernel Interface Error Codes”, for an explanation of these error codes.

1.5.4 Specific errors

Specific errors are those which are specific to a particular Kernel routine; the codes for such errors are listed in the routine header.

1.6 KI concepts

The following sections introduce Parasolid concepts that are specific to the use of the KI interface routines.

For further information on the concepts which apply also (or only) to the PK Interface see the ‘Parasolid Concepts’ chapter in the Parasolid Functional Description manual.

1.6.1 The world

The world has a tag value of 1, and it is a unique entry with its own type. It contains all parts (bodies and assemblies) in the session. As the world is unique it has no subtypes.

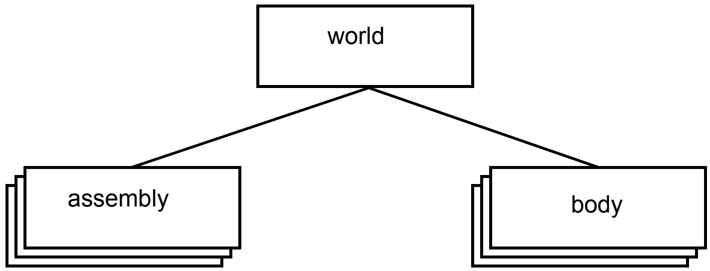


Figure 1–1 The world, assemblies and bodies

1.6.2 Lists

Lists provide a general method of structuring data, and are required or returned by many KI routines. They are typically used without an owner, but can be attach to a body (although they cannot be archived with the body). There are three types of list:

- Integer - holds integer values
- Real - holds (double precision) real values
- Tag - holds KI tags

1.6.3 Part states

The word part is used to mean “assembly or body”.

The ‘state’ of a part depends on whether and how, it has been archived during a session. You can find out the state of a part in the session using OUPART. The possible states are:

- **new** - Identified by the token ENSTNW. This is the initial state of a part which has been created in the current session, before it is archived.
- **stored** - (ENSTST) The state of a part which is an identical copy of a part in the archive. Archiving or retrieving a part sets its state to stored.
- **modified** - (ENSTMD). The state of a part which had stored state, but has been changed since.
- **unloaded** - (ENSTUN). The state of a part which was in memory earlier in the session, but has since been unloaded using UNLDPA. Alternatively, a part which is not loaded in the session, but is referred to by an assembly.
- **anonymous** - (ENSTAN). Anonymous parts are those which have not been archived in their own right, but because they are a component of an assembly.

SESTPA can be used to change the state of a part.

1.6.4 Archiving

Archiving is the process of saving parts which are in Parasolid's internal memory to external storage. The nature of this storage depends on your application's Frustrum.

Keys

Stored, modified and unloaded parts all have keys, which you can find by calling OUPART. New and anonymous parts do not have a key associated with them - they are said to be **unkeyed**.

The key is a text string which you will use to locate the archive data for that part. Your Frustrum uses the key either as a file-name, or an index into your application's database. The implementation of the key depends on the Frustrum. Parasolid remembers a part's key, during a session, but to retrieve the part in a later session you must also save the key somewhere.

It is possible to have several parts with the same key. This can occur if you save a part, change it and then retrieve it again from archive (this could happen several times). It can also occur if you retrieve an assembly which had components saved with it, when some of them are already in the modeler's memory.

Because of the limitations on SAVMOD and GETMOD, only one of the parts in memory with a particular key can be in 'stored' state. You can find a list of tags of parts which have the same key by calling the KI routine IDKYPA.

Transmitting

You call SAVMOD with the tag of a part and a key. You can only save 'new' or 'modified' parts. A 'part' is a body or an assembly - you cannot save subordinate entities, like edges and surfaces etc., on their own. Saving (or *transmitting*) a part changes its state to 'stored'.

When you archive a part, the following items are saved:

- All topological and geometric entities contained in the part.
- The user field of each entity.
- Node identifiers of all entities which have them (i.e. every type of topological/ geometric entity except fins, bodies and assemblies).
- Any groups belonging to the part.
- Any attributes belonging to entities in the part.
- Any construction geometry attached to the part (this can be done using DEFCON). Note that although you can attach lists to a part with DEFCON, they are not archived by SAVMOD.

See the following 'Assemblies' section for details of the extra information which is saved with assemblies.

Unloading

Once you have saved a part, you can unload it from memory using UNLDPA, which frees Parasolid's internal storage space for other models. The tag of the part remains valid, and if you need to use the part again later in the session, it will be reloaded automatically.

When you unload a part, you can still use the part's tag, key and box. Note that if the part is leaded again, the subordinate entities will probably not have the same tags as the ones they had before the part was unloaded.

Receiving

You retrieve a part from archive using GETMOD. Given the key of a part, this function obtains the archive data via the Frustrum, and re-loads the part into Parasolid's memory.

When you save a part, the data output through the Frustrum can be in one of two formats: text or binary. (Text format files are guaranteed to be portable between different machines; binary files can be in neutral (portable) or in machine speak format). There is an 'interface parameter' to control which of these formats the kernel is expecting to use. This parameter is identified to SEINTP by the token SLIPBT. If the format of the file you are trying to receive does not match that expected by the kernel, GETMOD will fail.

You can retrieve any part for which you have a key. It is your system's responsibility to ensure that Parasolid can find the archive data for the part. This is mostly dependent on the Frustrum. If the key cannot be found by the Frustrum (for instance because your implementation simply treats keys as filenames and has no information on which directory to look in) then GETMOD will fail.

GETMOD retrieves parts which were saved in a previous session and are as yet unknown in the current session, and it also works on 'unloaded' and 'modified' parts. The state of a part *after* you have retrieved it is 'stored'. If the part was 'modified' then the part is received as normal, resulting in two parts in memory which have the same key; one of which has state 'stored', and the other `modified'. If you try to retrieve (or receive) a part which is already in memory then GETMOD will fail with the error "Part already loaded", because the part is, by definition, already in internal memory.

When you are retrieving a part you must take care that the size of its user fields is the same as that set for your current session by STAMOD. If it is not, and you do not need to know the contents of the user field, you can prevent GETMOD from trying to read it by calling the routine SEINTP (set interface parameter) with the token SLIPUF, and value 0. This action will cause GETMOD to ignore the user fields which were saved with the part, create user fields of the length required by the current session, and set them to zero.

1.6.5 Assemblies

Assemblies allow the representation of collections of bodies as part of a single model. They can contain other assemblies as well as solids, generating a tree-like structure.

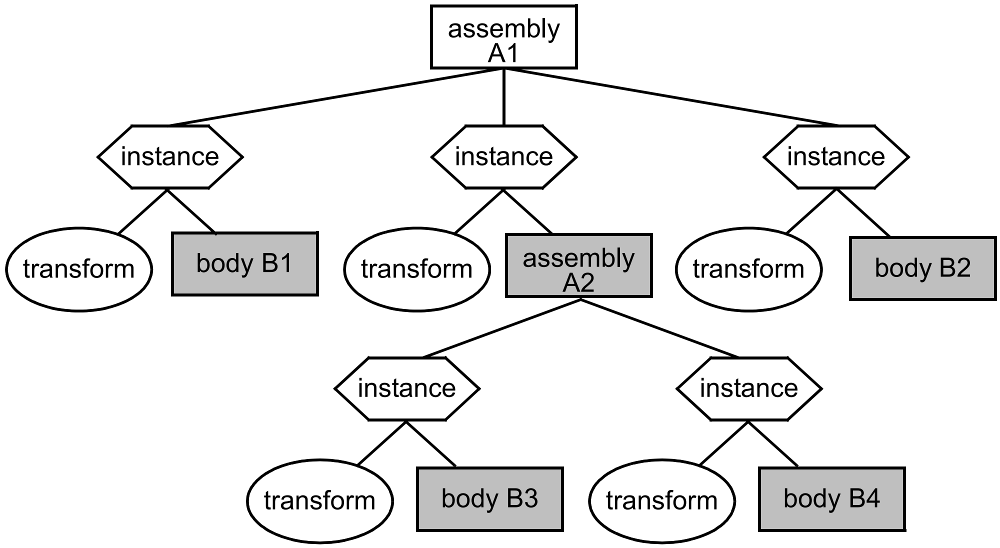


Figure 1–2 The instanced components of an assembly

Terminology

An **instance** is a pointer identifying a part, together with a definition of its location in the assembly (a **transformation**). A part is said to be **instanced** in an assembly if the assembly contains an instance somewhere which points to the part.

It is possible to instance the same body (or assembly) several times in an assembly, and because each new instance only points to the part rather than copying it, a lot of internal (and disk) space is saved.

The **components** or **sub-parts** of an assembly are all the parts which are instanced anywhere in the assembly. All the shaded parts (both bodies and assemblies) of Figure 1–2 are components of assembly A1. Similarly, the components of A2 are bodies B3 and B4.

The **occurrence** of a part within an assembly depends upon two things, which instance it belongs to and at what level. For example:

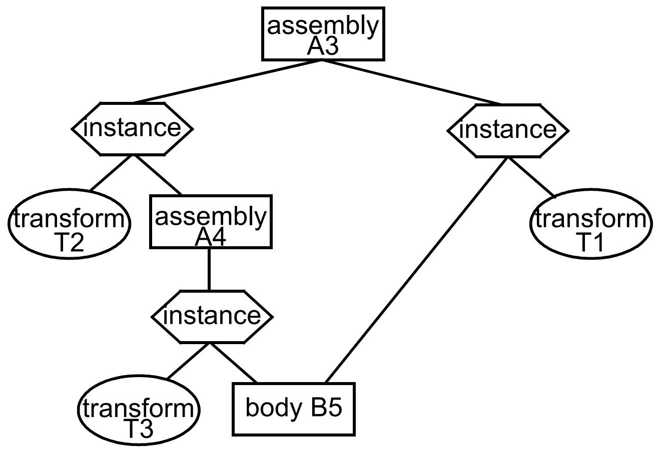


Figure 1–3 Two occurrences of one body in an assembly

In Figure 1–3there are two occurrences of body B5 in assembly A3. One is at the "top level" where it is instanced in its own right with a transformation T1. The second is as part of the top level instance of assembly A4. In this second occurrence, B5 is transformed twice: once because of the transform with it in A4 (T3) and then again by the transformation of A4 in A3 (T2).

Parasolid does not keep track of occurrences of parts in an assembly, you have to locate them yourself. This means that when performing operations on assemblies, such as drawing them, or allowing the users of your application to pick entities from them, if you want to be aware of which occurrences are which, you must program accordingly.

Creating assemblies

CREASS creates an (empty) assembly, to which instances can be added.

CREINS creates instances using the tag of the assembly, the tag of the part and the transformation (if any) to be applied to the part.

If you do not supply a transformation when you create an instance, it will not have a transformation attached to it, and the modeler will act as if the identity transformation is attached. Transformations can be changed using APPTRA.

1.6.6 Archiving and receiving assemblies

SAVMOD will archive an assembly and its components in one file (depending on their part states - e.g. ‘stored’ parts will not be archived again, the assembly file will contain a pointer to the stored part).

Alternatively, if SAVMOD is first used to archive each component of an assembly, the assembly “file” will simply contain pointers to the other files.

GETMOD will receive the archived assembly. The modeler works hard to maintain the consistency of the data structure in its internal memory, with respect to the state of all the parts. When the part you are receiving is an assembly, Parasolid also tries to link it up to any components which were saved separately from it. The success of these attempts depends on whether there are parts in memory with the same keys as parts you are trying to receive or instance, and if so, the state of those parts.

Rendering assemblies

You can create an image of an assembly by passing the tag of the assembly to any of the RRxxxx rendering routines. The data will be output through the GO in the usual way. Each segment will have its own body tag, but they will all have the same occurrence number. This refers to the occurrence of the assembly in the entity list passed to the RR routine, and not to assembly structure.

However, this will not enable you to relate any particular occurrence of a part in the assembly to its image, because there will be no links between the data output through the GO and the instances of parts in the assembly.

If you want to be able to identify a particular occurrence of an entity in the picture, then you should draw each component part of the assembly separately. Then when an entity comes out through the GO, as well as having the tag of the body, you can find out which instance it is by linking it back to the entity in the drawing list.

Processing KI assemblies using the PK interface

When a KI assembly is loaded into an application which uses only PK functions, PK_ASSEMBLY_ask_parts can be used to get the tags of its bodies (sub-assemblies need to be enquired recursively until all bodies are obtained). The bodies can then be archived as multiple bodies in a single part file, using PK_PART_transmit.

Many other PK functions which operate on a ‘part’ will accept either a body or an assembly as the input entity.

1.6.7 Senses

There are some differences between the PK and KI interfaces, in the way that senses are considered. The following is an explanation of the way senses have to be considered when using KI routines, where they differ from the PK concepts.

Surface sense flag

Every surface has an explicit sense flag. This indicates whether the orientation of the surface entity is the same as the natural orientation of the surface:

- If the surface sense is KI_true, the normal of the surface is the same as the natural one of the surface.
- If the surface sense is KI_false, it is opposite to the natural normal.

If a surface is negated by calling NEGENT, the surface normal is reversed. Since the orientation of a surface depends on the surface sense this has the effect of reversing the surface normal. For example, the orientation of simply curved surfaces would then point away from the concave side of the surface. The sense of a surface is found by calling OUTSUR.

Face reverse flag

Every face has a reverse flag associated with it which indicates whether the face normal is in the same direction as the surface normal, where the surface normal means the orientation of the surface taking into account the surface sense flag. The surface can be found by calling IDSOFF. If the face reverse flag is true the face normal is anti-parallel to the surface normal, and if the reverse flag is false the face normal is parallel to the surface normal.

To find the normal of a face first get the natural surface normal; then apply the surface sense; and finally apply the face reverse flag.

Face sense

Several KI routines refer to the 'face sense', therefore it is necessary to make the distinction between the face sense and the face reverse flag. In general terms, the face sense means the opposite of the face reverse flag, e.g. if the face sense is true then the face normal will be in the same direction as the surface normal, and if the face sense is false the face normal will be in the opposite direction to the surface normal.

The sense of a face can only be altered through the KI when a surface is attached to the face using ATTGEO or ATGETO, or when a face is tweaked to a surface using TWSUFA.

For consistency, the face sense must be correct. Therefore if the face sense is KI_true, the orientation of a surface must be in the same direction as the face normal, i.e. point outwards from the material of the body. If this is not the case, the face will either be concave instead of convex (or vice versa) or the face will be invalid and will fail the checks imposed by CHCKEN.

The following example shows a cylinder with a spherical face at one end. The surface normals of the spherical surface point away from the convex side of the surface.

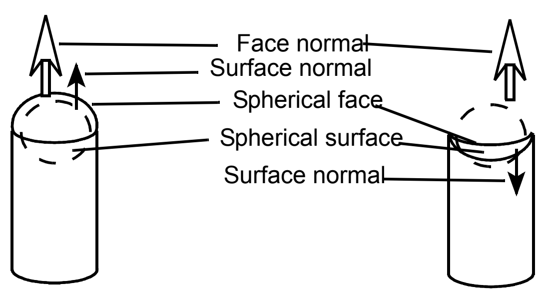


Figure 1–4 When the face sense is KI_true, the face is convex; when it is KI_false the face is concave

Curve directions

Curves directions and their relationship to other entities are considered differently in the KI, although the above conventions on loops, fins, edges and curves in the PK are also relevant.

- Curves have the same direction as the entities to which they are attached.
- The direction of a curve at a particular parameter may be found using ENPOPC and reversed by calling NEGENT.

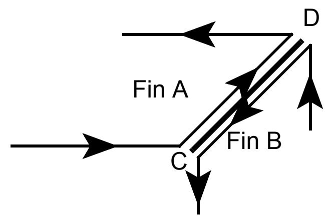


Figure 1–5 Fin and edge direction

- As can be seen from Figure 1–5 when fin A is the left fin of the edge the direction of the edge and any attached curve is from C to D. If the curve direction was reversed, the faces on each side of the edge would become inconsistent.
- When fin B is the left fin of the edge, the direction of the edge is from D to C and the attached curve must also have this direction if the adjacent faces are to be consistent.
- If the edge has no curve, but curves exist on the fins (this would occur, for example, during trimmed surface import), then the curve attached to fin A must go from C to D and if attached to fin B from D to C. If the direction of one curve is reversed, the face containing the owning fin would be invalid.
- Where a closed curve is attached to an edge or fin, see Figure 1–6 the relationship between the directions of the curve and owning edge/fin determines whether the edge/fin curves into or out of the face.

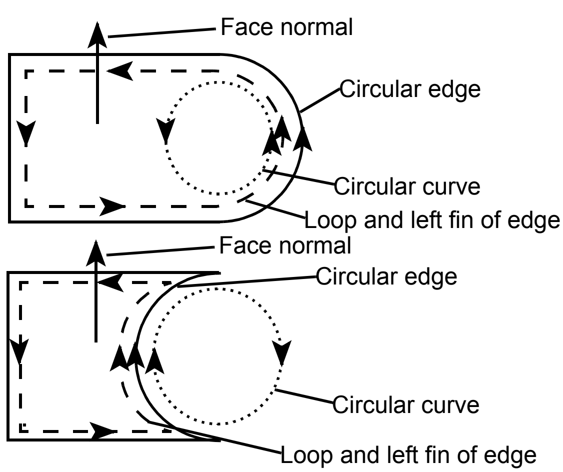


Figure 1–6 The relationship between the direction of the curve and its owning edge/fin

Trimmed curves

There is a difference in the parameterisation of curves (in particular trimmed curves) between the KI and the PK.

Consider a simple example of an infinite curve (i.e. parameterized between -infinity and infinity). Based on this curve is a trimmed curve which through the KI is seen as being parameterized between 0 and 10 (this could have been created by CRTRCU for instance). Through the PK, this curve is also seen as being parameterized between 0 and 10.

Now consider a trimmed curve based on the original curve and parameterized between 5 and 10. Through the PK, this curve is seen as being parameterized between 0 and 5. This convention of having a trimmed curve parameterized between 0 and X was introduced at the PK for STEP compliance.

Now take this trimmed curve from (2) and negate it using NEGENT:

- At the KI this will have the effect of negating the underlying curve, and the parameterisation of the trimmed curve will now run from -10 to -5 (-10 and -5 correspond to 10 and 5 of the curve before it was negated). The KI does not have negative sense trimmed curves, these are thought of as trimmed curves based on negative sense curves.
- At the PK this same trimmed curve will have parameterisation 0 to 5. As the PK has no concept of curve sense, the 0 and 5 correspond to the -5 and -10 of the negated curve (and the 5 and 10 of the original trimmed curve).

Thus in this case:

- the same answer will be obtained from both ENPOPC(curve, -10, 0) and PK_curve_eval(curve, 5, 0)
- PK_CURVE_ask_parm_different on this curve will return true
- PK_CURVE_convert_parm_to_pk(curve, -10) will return 5 as the PK equivalent to -10 in the KI parameterisation

Simultaneous use of ENPOPC and PK_CURVE_eval is not recommended. The use of PK_CURVE_eval only is recommended.

Kernel Interface Routines 2

2.1 Introduction

This chapter contains the specifications of all the Kernel Interface routines in alphabetical order. For the specifications of GO and Frustrum routines see Chapter 2, ‘Graphical Output Routines’ and Chapter 4, ‘Frustrum Routines’ of the Parasolid Downward Interfaces manual.

ADPAPE - Add parameter (line) to a B-curve or B-surface

Receives

```
KI_tag_geometry      *pg       B-curve or B-surface
KI_dbl_parameter     *t        split parameter
<KI_cod_papr>        *uorv     parameter in which to split
```

Returns

```
KI_cod_error            *ifail     failure indicator
```

Specific Errors

```
KI_bad_knots            new knot would cause invalid knot
                        multiplicity
KI_bad_parameter        parameter out of range
KI_bad_parametric_prop  unsuitable property
KI_wrong_entity         'pg' is not a B-curve or B-surface
KI_is_attached          geometric entity is attached to topology
```

Description This function modifies a B-curve or B-surface by inserting a knot, resulting in the addition of a (possibly null) segment, or row or column of patches. The shape of the B-curve or B-surface is not changed.

The parameter 'uorv' is not used if the entity is a B-curve. If the entity is a B-surface then 'uorv' must take one of the following values:

Token	Meaning
PAPRUP	add a parameter line in the u direction (i.e. a constant v pa rameter line)
PAPRVP	add a parameter line in the v direction (i.e. a constant u pa rameter line)

The parameter t at which the new knot is to be inserted must be within the range as given by ENCUPA (curves) or ENSUPA (surfaces) repectively.

The new knot must not increase the multiplicity of any existing knot to more than the order for an end knot, or order-1 for an internal knot. If 't' lies within a segment or patch then a new segment, row of patches ('uorv' = PAPRUP) or column of patches ('uorv' = PAPRVP) will be added to the curve or surface.

If 't' lies on a segment or patch boundary then the segment, row or column of patches added will be null. Adding a null segment to a curve or surface only affects the B-spline representation (it duplicates a knot).

A B-curve or B-surface cannot be modified if it is attached to an edge.

ADVXED - Adds a new vertex to a given edge

Receives

```
KI_tag_edge        *edge    to which vertex is added
KI_vec_position     point   position vector for new vertex
```

Returns

```
KI_tag_vertex          *newvrx       new vertex added
<KI_tag_edge>          *newedg       new edge added
KI_cod_error           *ifail        failure code
```

Specific Errors

```
KI_missing_geom    edge not geometrically specified
KI_not_on_edge     point not on edge curve
KI_coincident      point coincident with existing vertex
```

Description The routine adds a new vertex to a given edge, with vertex geometry to suit the supplied vector. Adding a vertex to a ring edge, e.g. a complete circular edge, will not split the edge into two, whereas other remaining cases will. Tags of the new vertex, and where appropriate the new edge, will be returned from the routine.

The routine will not allow the user to add a vertex to the edge where a vertex already exists, and will ensure that the new vertex lies within edge tolerance of the supplied edge.

APPTRA - Apply transformation

Receives

```
KI_tag_list_entity     *entity   entity or list of entities
KI_tag_transform       *transf   transformation to apply
```

Returns

```
KI_cod_error           *ifail    failure code
```

Specific Errors

```
KI_bad_sharing          sharing prevents transformation
KI_is_attached          geometric entity is attached to topology
KI_wrong_transf         invalid transformation; unsuitable entity for
                        transformation
KI_different_types      entities in list are not all of same type
KI_wrong_entity_in_list list contains entity of wrong type
```

Description Applies a transformation to an entity or list of entities. The types of entity permitted are:

- Assembly
- Instance
- Body
- Point
- Curve
- Surface
- Transformation

If the transformation is non-orthogonal (see CRETFM) it can only be applied to the following entities:

- B-surface
- B-curve
- foreign surface
- foreign curve
- SP curve on B-surface or foreign surface
- line
- trimmed curve of the above
- point
- transform
- body containing only the above

If a list of entities is given all the entities in the list must be of the same type.

The specified transformation is applied to the entity or entities as follows:

Entity	**Transformation applied to:**
Assembly	The transformation is applied to any instances and construction geometry in the assembly.
Instance	If the instance does not have a transformation attached a copy of the given transformation is attached to the instance. Otherwise, the trans formation is applied to the transformation attached to the instance (see below).
Body	The transformation is applied to all geometric entities in the body.
Point	The transformation is applied to the point. The point must not be attached to a vertex.
Curve	The transformation is applied to the curve. If the curve is an intersection curve its underlying surfaces are also transformed. The curve must not be attached to an edge (nor must any of its dependents be attached to any topology). Furthermore neither the curve nor any of its dependents may be shared with another geometric entity which is not a member of the list of entities being transformed.
Surface	The transformation is applied to the surface. Any dependent curves and surfaces are also transformed. The surface must not be attached to a face (nor its dependents, if present, to faces or edges.) Furthermore neither the surface nor any of its dependents may be shared with another geomet ric entity which is not a member of the list of entities being transformed.
Transformations	The two transformations are multiplied. If A and B are transformations, the result of applying A to B is a transformation which applies B and then A. The transformation being altered must not be attached to an instance.

Note: Only rotation and translation transformations can be applied to assemblies or instances.

ATGETO - Attach geometry to topology

Receives

```
int               *ntopol         number of topological entities
KI_tag_topology    topol[ntopol]  topological entities
KI_cod_logical     sense[ntopol]  face senses
int               *ngeom          number of topological entities
KI_tag_geometry    geom[ngeom]    geometric entities
```

Returns

```
KI_cod_error        *ifail         failure code
```

Specific Errors

```
KI_has_parent              'geom' is already attached
KI_bad_shared_dep          attempt to illegally share a dependent
                           of 'geom'
KI_bad_shared_entity       attempt to illegally share 'geom'
KI_geom_not_needed         'topol' already owns geometry
KI_geom_topol_mismatch     geometry/topology mismatch
KI_inconsistent_geom       inconsistent geometry
KI_wrong_entity            type of 'topol' incorrect
KI_invalid_geometry        geometry does not pass checks
KI_not_in_same_partition   entities are not all in the same partition
```

Description This function connects geometric entities (geom[]) to topological entities (topol[]). The array of senses (sense[]) must be the same length as the array of topology. The senses are used to set the face sense only and are ignored if the corresponding topology is not a face (edges and fins do not have a sense).

The length of the geometrical entity array ngeom must be either 1 or equal to the length of the topological entity array ntop.

If the geometrical entity array is the same length as the topological entity array then ATGETO connects each geometric entity to the corresponding topological entity in the arrays. A geometric entity may occur more than once in the array of geometrical entities and if this is the case then geometry will be shared, providing the sharing is legal.

If the geometrical entity array only has one element then this element will be connected to all of the elements in the topological entity array and thus shared between them.

Each geometric entity is connected to the corresponding topological entity if their types permit. The combinations of topology and geometry allowed are:

Geometry	**Topology**
surface	face
curve	edge
curve	fin
point	vertex
transform	instance

ATGETO will not copy any geometric entity, thus attachments are further restricted by constraints on the sharing of geometry:

- Geometry that will be shared after the call to ATGETO may only be shared by topology within one body. Points and transforms may not be shared.
- A curve cannot be shared between two edges which have opposing directions.
- Orphan geometry may be attached to a face, edge or vertex if neither the entity nor any of its dependents are attached to topology and neither the entity nor any of its dependents are shared with any other orphan entity.
- Construction geometry in a body may be attached to a face, edge or vertex so long as it is in the same body.

There are also restrictions on the types of curve that may be attached to edges and fins. Only trimmed curves with SP-curve basis curves may be attached to fins and the corresponding edge must be approximate. No curve may be attached to an approximate edge and no curves may be attached to the fins of an accurate edge.

When surfaces are attached to faces, 'sense' specifies the value to which the face sense is set. If 'sense' is true, the face normal is parallel to the surface normal. For all other combinations of geometry and topology the 'sense' is ignored.

The curve or surface must be capable of passing the checks imposed by CHCKEN.

The self intersection check is only performed if the appropriate option is set (see SEINTP).

If a transform is being attached to an instance it may only contain translation and rotation components. Reflections, scales and shears are not allowed.

> **Note:** ATGETO may not be used to attach construction geometry to a body or assembly: use DEFCON for this.

ATTGEO - Attach geometry to topology

Receives

```
KI_tag_topology          *topol      topological entity
KI_tag_geometry          *geom       geometric entity
KI_cod_logical           *sense      face sense
```

Returns

```
KI_cod_error             *ifail       failure code
```

Specific Errors

```
KI_has_parent            'geom' is already attached
KI_bad_shared_dep        attempt to illegally share a dependent of
                         'geom'
KI_bad_shared_entity     attempt to illegally share 'geom'
KI_geom_not_needed       'topol' already owns geometry
KI_geom_topol_mismatch   geometry/topology mismatch
KI_inconsistent_geom     inconsistent geometry
KI_wrong_entity          type of 'topol' incorrect
KI_invalid_geometry      geometry does not pass checks
KI_not_in_same_partition topol and geom are in different partitions
```

Description The geometric entity is connected to the topological entity if their types permit. The combinations of topology and geometry allowed are:

Geometry	Topology
surface	face
curve	edge
curve	fin
point	vertex
transform	instance

ATTGEO will not copy any geometric entity, thus attachments are further restricted by constraints on the sharing of geometry:

- A point or transform cannot be attached to more than one topological entity. A curve or surface may be shared by more than one topological entity, so long as they are in the same body.
- Orphan geometry may be attached to a face, edge or vertex if neither the entity nor any of its dependents are attached to topology and neither the entity nor any of its dependents are shared with any other orphan entity.
- Construction geometry in a body may be attached to a face, edge or vertex so long as it is in the same body.

There are also restrictions on the types of curve that may be attached to edges and fins. Only trimmed curves with SP-curve basis curves may be attached to fins and the corresponding edge must have a user defined tolerance. No curve may be attached to a toleranced edge and no curves may be attached to the fins of a non-toleranced edge.

If a transform is being attached to an instance it may only contain translation and rotation components. Reflections, scales and shears are not allowed.

If data checking is on, (see SEINTP and OUINTP), simple checks are made that geometry to be attached to a face, edge or vertex is geometrically consistent with any geometry attached to neighboring topological entities, but the checks do not guarantee that the body is valid. For example, if a surface is attached to a face, a check is made that the curves of the edges of the face lie on the surface, but no check is made that the surface does not intersect other faces of the body. If the checks fail, attachment will not take place.

If data checking is off, the only checks that will be performed are that the entity to be attached is of the correct type.

When surfaces are attached to faces, 'sense' specifies the value to which the face sense is set. If 'sense' is true, the face normal is parallel to the surface normal. For all other combinations of geometry and topology the 'sense' is ignored.

The curve or surface must be capable of passing the checks imposed by CHCKEN.

The composite geometry checks are only performed if SLIPCO (see SEINTP) is set to 0.

The self intersection check is only performed if SLIPSI (see SEINTP) is set to a non zero value.

> **Note:** ATTGEO may not be used to attach construction geometry to a body or assembly: use DEFCON for this.

BLECHK - Check the local validity of unfixed blends

Receives

```
KI_tag_list_edge          *edges     edges to check
int                       *full      level of checking
```

Returns

```
KI_int_nitems        *nerror number of invalid blends
<KI_tag_list_int>    *errors list of failure codes, the codes are
                             given below
<KI_tag_list_edge>   *erreds list of invalid edges
<KI_tag_list_entity> *topols list of tags associated with the failure
                             codes
KI_cod_error         *ifail  failure code
```

Specific Errors

```
KI_general_body          general body
KI_bad_spec_code         value of 'full' out of range
```

Description This is a diagnostic function, used to determine why BLEFIX failed, or whether an alteration in an unfixed blend has removed a problem.

For each edge in 'edges' that has a blend attribute attached the associated blending surface parameters are checked for consistency with geometry and blend attributes attached to neighbouring entities. Edges in 'edges' that have no blend attribute attached are ignored. If 'full' is 1, the blending surface parameters are checked for consistency at the vertices. If 'full' is 2, additional checks are made to ensure that the blend boundaries are legal. Even if BLECHK finds no inconsistencies, the blends are not guaranteed to fix; some problems can only be detected by BLEFIX making up the new faces.

'nerror' is the number of edges in 'edges' for which blending errors have been found. The list 'erreds' contains the tags of those edges for which errors were found and 'errors' and 'topols' are two parallel lists of failure codes and associated topology tags.

Some failure codes have no associated entity: in these cases the entry in 'topols' will be the null tag. The meanings of the failure codes in the 'errors' list and the corresponding types of entity in the 'topols' list are as follows:

Code	Explanation	Tag Type
BLCCTN	Blend on tangent edge is not legal.	-
BLCCSN	Blend ending on a surface singularity is not legaL.	TYTOVX
BLCCOT	Unsupported version 1 or 2 blend type.	-
BLCCMX	Configuration of edges at vertex too complex.	TYTOVX
BLCCRS	Edge being blended is on boundary of sheet.	-
BLCCRE	Edge being blended has non-manifold vertex	TYTOVX
BLCCTV	Blend ends on illegal two edge vertex.	TYTOVX
BLCCHM	Edge geometry unsuited to asymmetric chamfer ranges.	-

BLCCXT	Blend requires invalid extension of B-surface.	TYTOFA
BLCCIR	Range inconsistent with adjacent blended edge.	TYTOED
BLCCIT	Type of blend inconsistent with adjacent blended edge.	TYTOED
BLCCAB	Adjoining edge not blended.	TYTOED
BLCCOL	Blend completely overlaps edge loop.	-
BLCCOB	Overlapping blends.	-
BLCCOU	Unblended edge overlapped by blend.	TYTOED
BLCCUN	Unspecified numerical problem with blend.	-
BLCCUE	Undetermined problem at end of blend.	TYOTVX
BLCCRL	Range of blend on face too large.	TYTOFA
BLCCOE	Illegal overlap on end boundary.	TYTOVX
BLCCIE	Blend has illegal end boundary.	TYTOVX
BLCCIX	Cannot intersect chamfers at this vertex.	TYTOVX
BLCCEX	End boundary intersects unblended edge.	TYTOED
BLCCOI	Illegal blend on another edge prevented full check.	TYTOED
BLCCIP	Cliffedge blend range inconsistent with cliff edge.	-

This function is not supported for edges on general bodies.

BLECRB - Define a rolling ball blend or chamfer

Receives

```
KI_tag_list_edge   *edges            edges to be blended
int                *type             blend type
KI_dbl_distance    *range1           blend range 1
KI_dbl_distance    *range2           blend range 2
<KI_int_nitems>    *nprops           number of blend properties
KI_cod_blec         props[nprops]    array of blend properties
<KI_tag_list>       pvals[nprops]    array of tags of data lists
```

Returns

```
KI_tag_list_edge  *blend   list of edges to which blends have been
                           attached
int               *nblend  number of edges to which blends have been
                           attached
KI_cod_error      *ifail   failure code
```

Specific Errors

```
KI_bad_blend_param  blend parameter out of range; illegal blend property
KI_general_body     general body
```

Description Attaches unfixed blends with the given parameters to a list of edges. Any existing attribute is replaced by one with the given parameters.

```
'type' 1 => rolling ball blend
       2 => chamfer
```

The blend ranges are interpreted according to the following table.

Blend Type	`range1'	`range2'
Rolling ball	radius of rolling ball	ignored
Chamfer	range `on left-hand face'	other range

'range1' and 'range2' must be greater than twice the maximum tolerance of all edges in 'edges' and their associated vertices.

'props' consists of an array of 'nprops' blend properties, and 'pvals' consists of a parallel array of tags of data lists for those properties, which can frequently be null tags. The meanings of the properties and the associated data are given on the following page:

- BLECCL cliffedge blend - associated value in pvals: tag of list containing tag of edge at cliff top.

  Creates a blend tangent to the face adjacent to the blended edge and touching the edge at the 'top' of the cliff. This property is only valid if only one blend is being created with this call to BLECRB.

  This property is only valid for rolling ball blends.

- BLECPR propagate flag - associated value in pvals: null tag.

  Blend will be propagated over tangent edges, or past other unfixed blends if the resultant combination of blends at a vertex would be invalid, and blending the third edge results in a valid combination. The default behaviour is no propagation.

> **Note:** That if a blend does propagate, and you wish to remove the resulting blend attributes, BLEREM will need to be passed the tags of the edges propagated on to as well as that of the original edge.

  This property is invalid for cliff-edge blends.

- BLECTL set tolerance - associated value in pvals: real list containing tolerance.

  Sets the tolerance for the blended edge. This argument is only allowed when defining a chamfer surface. The default tolerance is 1000 * modeller resolution.

There are also some tokens relating to overflow behaviour.

The default overflow behaviour of a blend is as follows;

When fixing a blend, it is possible that the blend as defined by its basic parameters would lie outside the faces adjacent to the edge being blended. If this is so, the blend must 'overflow'.

If the configuration allows us, we do this by 'smoothly overflowing' and creating a blend one of whose underlying faces is the one it has overflowed onto.

If the configuration is unsuitable for this, we try to 'cliff overflow' - that is replace the appropriate section of the blend by a cliffedge blend running along the appropriate edges (Cliffedge overflows are only implemented for rolling ball blends).

If the configuration is unsuitable for this as well, we will then 'notch overflow' - that is, merely trim the blend by the other faces in the region, leaving its surface geometry unchanged. If the configuration is unsuitable for this as well, the blend will not fix.

By default neither cliffedge or smooth overflows will occur when the edge being overflowed is of the same convexity as the edge being blended.

By default also, cliffedge overflows will not occur at the ends of blends, except where they meet another blend smoothly.

Overflow Type	Prevent	Allow when Same Convexity	Allow at End
SMOOTH	BLECNS	BLECSM	*
CLIFFEDGE	BLECNC	BLECSC	BLECEC
NOTCH	BLECNN	*	*

A * means that this behaviour is always possible.

All these tokens have an associated value in pvals of a null tag. Note that while any combination of these tokens is allowed the preventative tokens will, of course, negate the efffects of the other overflow tokens.

- BLECRI ribs will be drawn by RRVIND - associated value in pvals: tag of real list containing ribspace.

  Where ribspace is the approximate separation of rib lines. (0 < ribspace < size)

- BLECDF draw and fix flag - associated value in pvals: null tag.

  Causes the blend to be ignored by BLEFIX, BLECHK and RRVIND, which will all treat the edges as unblended. Default behaviour is for an unfixed blend to be fixed, checked and drawn as required. Note that that an unfixed blend will only be drawn in RRVIND if BLECDF has not been set and RRVIND has been called with option RROPUB.

A successful result will produce a new blend attribute. It will return the tag of the edge(s) to which the blend attribute is attached. See BLEFIX for making the blends a part of the topology.

> **Note:** An unfixed cliff-edge blend will be invalidated if stored between sessions.

This function is not supported for edges on general bodies.

BLECVR - Define a variable radius blend

Receives

```
KI_tag_edge      *edge              edge to be blended
KI_int_nitems    *npts              number of data points
KI_vec_position   points [npts]     data points
double            values [npts*3]   data values
<KI_int_nitems>  *nprops            number of blend properties
KI_cod_blec       props[nprops]     array of blend properties
<KI_tag_list>     pvals[nprops]     array of tags of data lists
```

Returns

KI_tag_list_edge	*blends	list of edges to which blends have been attached
int	*nblend	number of edges to which blends have been attached
KI_cod_error	*ifail	failure code

Specific Errors

KI_bad_blend_param	blend parameter out of range illegal blend property
KI_general_body	general body

Description Attaches a variable radius blend with the given parameters to an edge. Any existing attribute is replaced by one with the given parameters.

'npts' is the number of data points on the edge at which the blend ranges and rho value are defined. 'points' is an array containing the positions of the data points and 'values' is an array of the associated values. These must be given in the order: range of blend off left face at first point, range of blend off right face at first point, rho value at first point, then the values for subsequent points in the same order. All ranges must be greater than twice the maximum tolerance of the edge and its associated vertices.

> **Note:** Currently the two ranges must be equal and the rho value zero at each data point.

'props' consists of an array of blend property tokens, of length 'nprops', and 'pvals' consists of a parallel array of tags of lists of values - for several properties this can be a null tag.

- BLECTL set tolerance - associated value in pvals: real list containing tolerance.

  Sets the tolerance for the blended edge. The default tolerance is 1000 * modeler resolution.

- BLECCR circular cross-section - associated value in pvals: null tag.

  Forces the blend to have circular cross-section at all points. The two ranges must be equal and the rho value zero at each data point. This property is currently default behavior and does not need to be set.

- BLECLI linear radius variation - associated value in pvals: null tag.

  Causes the blend ranges to vary naturally. The default behavior is for the range variation to be constrained in such a way as to ensure that they would meet other blends smoothly at the vertices.

- BLECPR propagate flag - associated value in pvals: null tag.

  Blend will be propagated over tangent edges, or past other unfixed blends if the resultant combination of blends at a vertex would be invalid, and blending the third edge results in a valid combination. The propagation will consist of exact rolling ball blends.

> **Note:** If a blend does not propagate, and you wish to remove the resulting blend attributes, BLEREM will need to be passed the tags of the edges propagated on to as welll as that of the original edge.

The default behavior is no propagation.

- BLECDF draw and fix flag - associated value in pvals: null tag.

  Causes the bledn to be ignored by BLEFIX, BLECHK and RRVIND, which will all treat the edge as unblended.

> **Note:** An unfixed blend will only be drawn in RRVIND if RRVIND has been called with option RROPUB, and this property is not set.

There are also some tokens relating to overflow behaviour.

The default overflow behaviour of a blend is as follows;

When fixing a blend, it is possible that the blend as defined by its basic parameters would lie outside the faces adjacent to the edge being blended. If this is so, the blend must 'overflow'.

If the configuration allows us, we do this by 'smoothly overflowing' and creating a blend one of whose underlying faces is the one it has overflowed onto.

If the configuration is unsuitable for this, we try to 'cliff overflow' - that is replace the appropriate section of the blend by a cliffedge blend running along the appropriate edges.

If the configuration is unsuitable for this as well, we will then 'notch overflow' - that is, merely trim the blend by the other faces in the region, leaving its surface geometry unchanged. If the configuration is unsuitable for this as well, the blend will not fix.

By default neither cliffedge or smooth overflows will occur when the edge being overflowed is of the same convexity as the edge being blended.

By default also, cliffedge overflows will not occur at the ends of blends, except where they meet another blend smoothly.

Overflow Type	Prevent	Allow when Same Convexity	Allow at end
SMOOTH	BLECNS	BLECSM	*
CLIFFEDGE	BLECNC	BLECSC	BLECEC
NOTCH	BLECNN	*	*

A * means that this behaviour is always possible.

All these tokens have an associated value in pvals of a null tag. Note that while any combination of these tokens is allowed the preventative tokens will, of course, negate the effects of the other overflow tokens. For further explanation, and examples, of the use of these tokens, please refer to the blending section of the manual.

A successful result will produce a new blend attribute. It will return the tag of the edge(s) to which the blend attribute is attached.

See BLEFIX for making the blends a part of the topology.

This function is not supported for edges on general bodies.

BLEENQ - Enquire blend parameters.

Receives

```
KI_tag_edge           *edge     edge whose blend is required
```

Returns

```
KI_tag_face           *face1    face to left of edge
KI_tag_face           *face2    face to right of edge
int                   *type     type of blend
double                *range1   range of blend on face1
double                *range2   range of blend on face2
<KI_int_nitems>       *nprops   number of blend properties
<KI_tag_list_int>     *props    blend properties
<KI_tag_list_list>    *pvals    blend property values
KI_cod_error          *ifail    failure code
```

Description The parameters of the blending surface defined on 'edge' are returned by this routine. It can also be used to find if an edge is blended.

The returned arguments are as follows:

Argument	Meaning
`face1'	The face to the left of the edge.
`face2'	The face to the right of the edge.
`type'	0=> no blend on edge 1=> exact rolling ball blend 2=> exact chamfer 3=> variable radius rolling ball
`range1'	The range of the blend associated with `face1'.
`range2'	The other range of the blend

In the case of a variable radius blend, this function will only return the end radii of the blend. 'range1' will refer to the range at the start of the edge 'range2' will refer to the range at the end of the edge. The arguments 'props' and 'pvals' consist of two parallel lists of blend properties and associated data, each of length 'nprops', such as would be passed to BLECRB or BLECVR.

BLEFIX - Fix blends in a body

Receives

```
KI_tag_body          *body    body to fix blends in
```

Returns

```
KI_int_nitems        *nblend  number of blend faces
<KI_tag_list_face>   *blends  list of created blend faces
<KI_tag_list_list>   *faces   list of underlying faces
<KI_tag_list_int>    *edges   list of (dead) tags of edges
```

```
KI_cod_blcc           *error    first error from blending body
<KI_tag_edge>         *err_ed   edge associated with error
<KI_tag_topology>     *topol    topology associated with error
KI_cod_error          *ifail    failure code
```

Specific Errors

```
KI_cant_fix_blends         could not fix blends in body
KI_general_body            general body
```

Description Any edges of the body which have blend attributes are changed into full faces with the appropriate blending surface geometry.

BLEFIX returns the number of created blend faces, and three parallel lists containing the created blend face tags, lists of tags of the associated underlying faces of the blend faces, and the tags of the edges which were blended to produce this set of blend faces.

'blends' will contain 'nblend' faces, which are the created blend faces.

'faces' will contain 'nblend' lists of faces. These are the underlying faces associated with the blends. The underlying faces of a blend are the faces whose geometry determines the blend surface geometry. Some of the face tags may be null. This occurs if there is no face in the final body with the appropriate surface geometry, which can be the case if a face has been entirely blended away, or for the cliff side of a cliffedge blend.

'edges' will contain 'nblend' integers. These integers are the (dead) tags of the edges which were blended to produce this set of blends. The same value may occur several times in this list if, for example, a blend has overflowed, since all the corresponding faces in 'blends' were caused by the same blend attribute.

> **Note:** The number of tags in each list of underlying faces is the number of original underlying faces for that blend, but that the tags returned are those of the faces after the blend has been fixed, and so some of them may be null (if the face has been completely blended out).

If the attempt to fix the blend fails due to an error which would be detected by BLECHK, the edge associated with the error will be returned as 'err_ed' and 'error' and 'topol' are the same as would be returned if BLECHK was called with 'err_ed'. These can be used to locate the reason for failure to fix the blend. Note that only the first error will be returned. If no errors are found 'error' will be zero and 'err_ed' and 'topol' will be null tags.

> **Note:** Blends created with the property BLECDF will not be fixed.

This function is not supported for general bodies.

BLEFXF - Create a blend between the specified faces

Receives

```
KI_tag_list_face     *l_wall          list of faces in left wall
KI_tag_list_face     *r_wall          list of faces in right wall
KI_cod_logical       *l_rev           blend direction from left wall
```

```
KI_cod_logical       *r_rev            blend direction from right wall
<KI_int_nitems>      *ntokens          number of blending tokens
KI_cod_fxft           tokens[ntokens]  blending tokens
<KI_tag_list>         bdata[ntokens]   data associated with tokens
```

Returns

```
KI_cod_fxfe          *status           blend success status flag
<KI_tag_list_entity> *s_data           data associated with status flag
<KI_tag_list_body>   *sheets           the blend sheets created
<KI_int_nitems>      *nblend           number of blend faces created
<KI_tag_list_face>   *blends           the blend faces
<KI_tag_list_list>   *unders           list of lists of underlying data
KI_cod_error         *ifail            failure code
```

Specific Errors

```
KI_fxf_blend_failed       could not create blends on body
KI_fxf_blend_bad_token    Illegal face face blend token
KI_not_in_same_partition  faces are not all in the same partition
```

Description This function creates one or more blends between the two sets of faces provided. The user specifies whether the blend will be attached to the bodies involved or returned as a separate sheet body. For further details on this function, see the blending section in the Functional Description.

'l_wall' is a list of the faces which make up the 'left' wall. These faces must all lie in the same shell of the same body.

'r_wall' is a list of the faces which make up the 'right' wall. These faces must all lie in the same shell of the same body.

The two walls do not need to lie in the same body. If the blend does not contact either the first face in 'l_wall' or the first face in 'r_wall' (or both), then BLEFXF will fail.

'l_rev' is a logical determining which side of 'l_wall' the blend lies. The blend lies in the direction of the face normals if 'l_rev' is false.

'r_rev' is a logical determining which side of 'r_wall' the blend lies. The blend lies in the direction of the face normals if 'r_rev' is false.

'tokens' consists of 'ntokens' face face blend properties, and 'bdata' is a parallel array of 'ntokens' tags of data lists for these properties, some of which may be null tags. The meanings of the various tokens are given below.

Trimming Tokens The user may give more than one trimming token to specify attachment and trim options.

- FXFTAT Trim blend and walls and attach blend.

  The associated value in 'bdata' is the null tag.

  The blend end boundaries will be determined by the wall boundaries unless further trim options are specified.

  The walls will be trimmed by the blend boundaries.

  The blend will be attached to the walls. If the walls lie in different bodies the bodies will be combined, with the body containing the 'left' wall being considered as the target body. If 'l_rev' and 'r_rev' are not the same, the body containing the right wall will be negated in order to produce a consistent result.

This is the default trimming option.

- FXFTTW   Trim blend and walls.

  The associated value in 'bdata' is the null tag.

  The blend end boundaries will be determined by the wall boundaries.

  The walls will be trimmed by the blend boundaries.

  The blend will not be attached to the walls but will be created as a separate sheet body.

  This token is only valid if the blend is being formed between two sheet bodies.

- FXFTTB   Trim blend to walls.

  The associated value in 'bdata' is the null tag.

  The blend end boundaries will be determined by the wall boundaries.

  Unless walls are set to attach, FXFTAT, then walls will not be trimmed, and the blend will not be attached to the walls but wil be created as a separate sheet body.

- FXFTST   Short trim to walls

  The associated value in `bdata' is the null tag.

  The blend end boundaries will be the constant parameter lines determined by the wall boundaries such that the blend is as short as possible.

  Unless walls are set to attach, FXFTAT, then walls will not be trimmed, and the blend will not be attached to the walls but wil be created as a separate sheet body.

- FXFTLT Long trim to walls

  The associated value in `bdata' is the null tag.

  The blend end boundaries will be the constant parameter lines determined by the wall boundaries such that the blend is as long as possible.

  Unless walls are set to attach, FXFTAT, then walls will not be trimmed, and the blend will not be attached to the walls but wil be created as a separate sheet body.

- FXFTNT   Do not trim blend.

  The associated value in 'bdata' is the null tag.

  The blend will not be trimmed to the wall extent. Its end boundaries will be constant parameter lines.

  Unless walls are set to attach, FXFTAT, then walls will not be trimmed, and the blend will not be attached to the walls but wil be created as a separate sheet body.

Blend Definition Tokens The user may give any number of blend definition tokens, though some are mutually exclusive. The user must provide one of FXFTCB, FXFTVB, FXFTDB, FXFTHL or FXFTCL.

The table below gives the rules for combining tokens. Each column shows which of the other tokens:

`M` must be given,

‘O` can be given,

`X` must not be given

if the token at the head of the column is given.

	FXFTCB	**FXFTVB**	**FXFTHL**	**FXFTCE**	**FXFTTL**	**FXFTRC**	**FXFTCL**	**FXFTCC**	**FXFTDB**
FXFTCB	-	X	O	O	O	X	O	X	X
FXFTVB	X	-	O	O	O	M	O	X	X
FXFTHL	O	O	-	O	O	X	O	X	X
FXFTCE	O	O	O	-	O	O	O	X	O
FXFTTL	O	M	M	O	-	M	M	M	M
FXFTRC	X	O	X	O	O	-	X	X	X
FXFTCL	O	O	O	O	O	X	-	M	X
FXFTCC	X	X	X	X	O	X	O	-	X
TXTTDB	X	X	X	O	O	X	X	X	-

- FXFTCB Constant radius rolling ball blend.

  The associated value in 'bdata' is a list containing a positive double.

  The blend will be a constant radius rolling ball blend, whose radius is the given double.

- FXFTVB Variable radius rolling ball blend.

  The associated value in 'bdata' is a list containing two curve tags.

  The blend will be a variable radius blend. The first curve provided will be used as a parameter curve and the second as a law curve.

  The law curve is a B-curve whose parameter space is that of a portion of the parameter curve in the region of the blend. The law should be three dimensional though it does not represent 3-space points. The meaning of the coordinates are:

  x-coordinate = range of blend off left wall.

  y-coordinate = range of blend off right wall.

  z-coordinate = rho value.

  The ranges must be positive, except at the ends of the blend where they are allowed to be zero. The left and right ranges need not be equal unless: i) either range is zero, at which point both ranges must be zero, or ii) the token FXFTHL is also given.

  By default, the blend will have circular cross-section if the ranges are equal, and elliptical cross-section otherwise. The rho values of the law curve can be used to control the cross-sectional shape of the blend by providing the token FXFTRC (see below).

- FXFTHL Blend is constrained by tangent hold lines.

  The associated value in 'bdata' is a list of edge tags.

  The blend will be constrained to be a tangent hold line blend in the region of the given edges. If this token is provided with FXFTCB or FXFTVB the blend need not be constrained by these edges. If this token is given with FXFTVB, the left and right ranges of the law curve must be equal.

- FXFTCL Blend is constrained by conic hold lines.

  The associated value in 'bdata' is a list of edge tags.

  The blend will be constrained to be a conic hold line blend in the region of the given edges. If this token is provided with FXFTCB or FXFTVB the blend need not be constrained by these edges. If this token is given with FXFTCC or without FXFTCB or FXFTVB, then FXFTHL and FXFTCE must not be provided, and the number of edge tags in bdata must be at least two.

- FXFTCC Curvature Continuous blend.

  The associated value in 'bdata' is a list containing two curve tags.

  This token must be provided with the token FXFTCL and the blend will be constrained to be curvature continuous at the hold lines.

  The first curve provided in 'bdata' will be used as a parameter curve and the second as a depth control curve.

  The depth curve is a B-curve whose parameter space is that of a portion of the parameter curve in the region of the blend. The depth curve should be three dimensional though it does not represent 3-space points. The meaning of the coordinates are:

x-coordinate	A value strictly between 0 and 1 giving the position of the maximum depth of the blend between the blend walls. A value close to 0 will force the cross section to have maximum depth close to the left wall, whilst a value close to 1 will force the cross section to have maximum depth close to the right wall.
y-coordinate	A value strictly between 0 and 1 giving the maximum depth of the blend. A value close to 0 will force the maximum depth of the blend to be as shallow as possible, whilst a value close to 1 will force the blend to be as deep as possible.
z-coordinate	ignored.

- FXFTCE Blend is constrained by cliff edges.

  The associated value in 'bdata' is a list of edge tags.

  The blend will be constrained to be cliffedge blend in the region of the given edges.

- FXFTTL A tolerance is associated with this blend.

  The associated value in 'bdata' is a list containing the tolerance.

- FXFTRC Variable radius blend cross-section is controlled by rho values from the law curve.

  The associated value in 'bdata' is the null tag.

  The rho values must lie in the range 0 < rho < 1. The blend cross-section will be:

  1) an ellipse for 0 < rho < 0.5

  2) a parabola for rho == 0.5

3) a hyperbola for 0.5 < rho < 1.

- FXFTDB Variable radius disc blend.

  The associated value in 'bdata' is a list containing two curve tags. The first curve provided will be used as a parameter curve and the second as a law curve.

  The blend will be a variable radius disc blend. The generating disc lies in the normal plane of the parameter curve; its size is defined by the law curve.

  For description of the law curve, see the documentation of FXFTVB above. Note that the rho value is ignored, since this token is incompatible with FXFTRC.

  This token may not be provided with FXFTPR.

Control Tokens The user may supply any combination of sheet control tokens

- FXFTHP Help point provided.

  The associated value in bdata is a list of three doubles.

  The three doubles will be used as coordinates of a help point, which will be used to differentiate between multiple alternative possible blends between the walls. If there is a blend which passes close to the help point, it will be the one created.

- FXFTMS Create multiple blends sheets.

  The associated value in bdata is the null tag.

  BLEFXF will only create one blend unless this token is provided, in which case it will create all blends which lie in at least one of the master faces.

- FXFTPR Propagate blends.

  The associated value in bdata is the null tag.

  If this token is given the blends will be allowed to propagate past smooth edges beyond the provided walls.

- FXFTLP Limit Plane.

  The associated value in bdata is a list containing one or two tags.

  The tags must be the tags of planes.

  The blend will be trimmed to end in a constant parameter line determined by a plane. The blend will lie on the positive side of the plane.

  This token is only valid if one of the trimming tokens FXFTNT, FXFTTB, FXFTST or FXFTLT is also provided.

- FXFTEO End Overflow

  The associated value in bdata is the null tag.

  If this token is given the blend will be allowed to extend past notches, whether in the middle or at the end of the blend.

- FXFTSO Create solid body if possible

  The associated value in bdata is the null tag.

  If this token is supplied and the result of the blending operation encloses a volume then a solid body will be created. Otherwise a sheet body will be returned.

This token is only valid if the blend is being attached, i.e. the trimming token FXFTAT is used.

'status' is a success status flag for BLEFXF, for those cases where an ifail is not returned. 's_data' consists of supporting data. The possible returns are

- FXFEOK The requested blend succeeded.

  The associated value in 's_data' is the null tag.
- FXFEST The blend has not been attached, nor have the walls been trimmed but blend sheet bodies have been created. This token is only returned if BLEFXF was called with FXFTAT or FXFTTW and represents a partial success.

  The associated value in 's_data' is the tag of a list. This list contains the tag of a list of three doubles and the tag of some topology. These indicate the region where the blend could not be attached.
- FXFEER The blend could not be created.

  The associated value in 's_data' is the null tag.
- FXFEID The user has supplied insufficient data to define a blend.

  The associated value in 's_data' is the null tag.
- FXFEXD The user has supplied inconsistent data.

  The associated value in 's_data' is the null tag.
- FXFEIF The user has supplied an invalid wall of faces.

  The associated value in 's_data' is the invalid list of faces.
- FXFEIR The user has supplied an invalid range definition.

  The associated value in 's_data' is the null tag.
- FXFEIH The user has supplied invalid tangent hold line data.

  The associated value in 's_data' is the invalid tangent hold line data.
- FXFEIC The user has supplied invalid cliff edge data.

  The associated value in 's_data' is the invalid cliff edge data.
- FXFEFC A face in a wall is too tightly curved for the blend to fit.

  The associated value in `s_data' is the face in question.
- FXFERS The blend range is too small.

  The associated value in `s_data' is a list containing one double. This is the suggested larger range for the blend.
- FXFERL The blend range is too large.

  The associated value in `s_data' is a list containing one double. This is the suggested smaller range for the blend.
- FXFELN `l_rev' is incorrect.

  The associated value in `s_data' is the null tag.
- FXFERN `r_rev' is incorrect.

The associated value in `s_data' is the null tag.

- FXFEBN Both `l_rev' and `r_rev' are incorrect.

  The associated value in `s_data' is the null tag.

- FEFESC The blend sheet(s) intersect one another and hence could not be attached.

  The associated value in `s_data' is the tag of a list. This list contains the tag of a list of three doubles and the tags of some topology. These indicate which sheet(s) clashed and the point at which they clashed.

- FEFEWC The blend, which has been attached, has combined two bodies and has produced a face-face inconsistency elsewhere in the model.

  The associated value in `s_data' is a list containing the tags of a pair of clashing faces.

- FXFEFF The blend, which has been attached, has caused a face-face inconsistency.

  The associated value in `s_data' is a list containing the tags of a pair of clashing faces.

- FXFEGX The blend contains face(s) with self intersecting geometry. The associated value in `s_data' is a list containing the tags of the blend faces with self-intersecting surfaces.

- FXFERV The user has supplied invalid rho values in the law curve.

  The associated value in 's_data' is the null tag.

- FXFEAR The user has supplied asymmetric ranges which are inconsistant with the underlying geometry.

  The associated value in 's_data' is the null tag.

- FXFECL The user has supplied invalid conic hold line data.

  The associated value in 's_data' is the invalid conic hold line data.

- FXFEIS The user has supplied an invalid parameter spine.

  The associated value in 's_data' is a lsu containing the tag of the invalid spine and possibly a list of three doubles, representing a point where the spine is unsuitable.

If BLEFXF has successfully created a blend, then 'sheets' will consist of a list of all new blend sheet bodies created, or null if the blend has been attached. `blends' will consist of a list of the `nblend' blend faces. 'unders' will consist of a parallel list of 'nblend' lists of integers. These lists contain the (possibly dead) tags of the underlying topology used to determine the equivalent blend face in 'blends'. Each list will consist of the tag of the face in the left wall, then the tag of the face in the right wall, then any other data that determined the geometry of the blend face, such as a cliff edge or tangent hold line.

BLEREM - Remove blend attributes

Receives

```
KI_tag_list_edge   *edges  list of edges to remove blends from
```

Returns

```
KI_cod_error       *ifail  failure code
```

Description Any blend attributes are removed from the edges.

BLNAFF - Find edges and faces affected by blend

Receives

```
KI_tag_edge          *edge    edge
```

Returns

```
KI_tag_list_edge     *iedge   list of affected edges
KI_int_nitems        *nedge   number of edges in 'iedge'
KI_tag_list_face     *iface   list of affected faces
KI_int_nitems        *nface   number of faces in 'iface'
KI_cod_error         *ifail   failure code
```

Specific Errors

```
KI_bad_blend_bound            illegal blend boundary
KI_not_blended                no blend on edge
KI_general_body               general body
```

Description Finds lists of edges and faces that are affected by the blending surface on the given edge. The faces in 'iface' and edges in 'iedge' are those partially or completely overlapped by the blending surface.

The list in 'iedge' will always contain the original edge 'edge'.

This function is not supported for edges on general bodies.

BLNDVX - Blends vertices on sheets and wires

Receives

```
KI_tag_list_vertex     *vertex   vertex or vertices to blend
KI_dbl_distance        *radius   blend radius
```

Returns

```
KI_tag_list_edge       *neweds   list of new edges
KI_tag_list_vertex     *newvxs   list of new vertices
KI_cod_error           *ifail    failure code
```

Specific Errors

```
KI_wrong_surface            surfaces not suitable
KI_invalid_bodies           invalid body
KI_edges_intersect          blends would intersect
KI_blends_overlap           blends would overlap
KI_cant_blend_vertex        blending can't be done
KI_wrong_number_edges       not two edges at vertex
KI_solid_body               vertices on solid body
KI_different_bodies         vertices on different bodies
KI_general_body             general body
```

Description Blends vertices from sheet or wire bodies with the given radius.

Only two edges should meet at the vertex, and these two edges must share a common plane. If the face between the two edges has a surface then it must be planar. A circular blend is created between the two edges.

All vertices in the list should belong to the same body.

If the two edges at the vertex are parallel so that the profile of the body is smooth across the vertex then that vertex will be ignored. If blending is not possible for any one of the vertices in the list for any other reason then none of the blending will be done.

If the body is planar then a valid body will be returned. If the body is non-planar then blending may create a self-intersecting body. In this case if local checking is on then the function will return an ifail stating that the body is invalid.

If any loop on a face is excluded from the face by a blend then this loop will be removed from the body.

Lists of the blend edges and the new vertices are returned.

This function is not supported for vertices on general bodies.

BOPBYS - Global or local boolean operation on bodies

Receives

```
KI_tag_list_entity   *target         target body or list of faces
KI_tag_list_entity   *tools          tool bodies or faces of tool body
<KI_int_nitems>      *nopts          number of boolean options
KI_cod_boop           opts[nopts]    boolean option codes
<KI_tag_list_entity>  optdta[nopts]  boolean option data lists
```

Returns

```
KI_tag_list_body     *bodys          resulting bodies
KI_int_nitems        *nbodys         number of bodies
KI_cod_error         *ifail          failure code
```

Specific Errors

```
KI_partial_no_intersect   No imprinting in local boolean
KI_boolean_failure        Inconsistent arguments, or internal error
KI_non_manifold           Non-manifold result
KI_t_sheet                T-sheet
KI_partial_coi_found      Boolean failure due to partial coincidence
KI_cant_intsc_solid_sheet Cant intersect solid target with sheet tool
                          bodies
KI_solid_has_void         Illegal void
KI_not_solid              Body is not solid
KI_not_sheet              Body is not sheet
KI_opposed_sheets         Attempt to unite opposed sheets
KI_cant_unite_solid_sheet Attempt to unite solid and sheet
KI_unsuitable_topology    A faceset selector is from boundary or wrong
                          body
KI_tool_faces_many_bodies Tool faces are from more than one body
KI_targ_faces_many_bodies Target faces are from more than one body
KI_mixed_sheets_solids    Mixture of sheet and solid tool bodies
KI_instanced_tools        Instanced tool bodies
KI_duplicate_targets      Duplication in list of target faces
KI_duplicate_tools        Duplication in list of tool bodies
KI_too_many_targets       Too many target bodies
KI_unsuitable_entity      Target or tool not a face or body
KI_wire_body              Target or tool has wireframe or acorn
                          components
```

KI_missing_geom	Target or tool has incomplete geometry
KI_same_tool_and_target	Tool body is also target body
KI_contradictory_request	Bad combination of options or data for type of boolean
KI_not_in_same_partition	Target and tools are not all in the same partition
KI_general_body	Target or tool is general body

Description **Introduction:** A boolean operation between the target body and the tool bodies is performed. The target body (workpiece) is modified, the tool bodies modifiers) are deleted, and their tags become dead. The resulting body or bodies replace the workpiece in the world, and are returned in a list.

The boolean may be global or local. A global boolean involves the comparison of all face pairs from the target and tool bodies, and so can guarantee topological consistency in the result.

A local boolean is faster than a global boolean as it involves the comparison of only the face pairs given in 'target' and 'tools' but topological consistency in the result cannot be guaranteed.

This function will accept general bodies in 'target' or 'tools' only if the interface parameter SLIPGT is set to one (see SEINTP).

The Boolean Algorithm: A local or global boolean may be a union, subtraction, intersection, or trim with sheet. The type of boolean performed can be selected with an option code in 'opts' (see below). If no option code is supplied, the default action is union.

Booleans in Parasolid may be thought of as being performed in three main phases. These are imprinting, gluing, and selection.

In the imprinting phase the faces of the target and the tool bodies are intersected with each other to produce new edges where they meet. These edges divide the faces of each body into facesets which are either inside, outside, or on the boundary of, the other bodies.

In the gluing phase the resulting sets of faces are joined together into a single intermediate body.

In the selection phase the parts of the model which are to be kept or rejected are selected, according to the type of boolean being performed and the options supplied, using information gained in the earlier phases.

Booleans Performed with SLIPGT set to zero: If SLIPGT is set to zero, the result of the boolean may be any number of bodies, each of which is a manifold and connected solid or sheet.

A union extends the target body by gluing to it all facesets of the tool bodies which are outside the target body.

A subtraction modifies the target body by removing all facesets which overlap with the tool bodies.

An intersection reduces the target body to only those facesets which overlap with the tool bodies.

Trim with sheet reduces the target to only those facesets behind the sheet tool bodies.

Punch is used in conjunction with union or subtraction. It takes a sheet target and solid tools to give a sheet result. Punch modifies a sheet target body by removing all facesets which overlap with the solid tool bodies and gluing to it all facesets of the tool bodies which are in front of (for unions) or behind (for subtractions) the sheet target body.

Booleans Performed with SLIPGT set to one: With SLIPGT set to one, the result of the boolean will be a single body, which may be disconnected, non-manifold, of mixed dimension, or any combination of these. The options BOOPPS and BOOPPV may be used to 'prune off' lower-dimension components of the result (see below).

A union extends the target body after imprinting to include all the facesets from both 'tools' and 'target'.

A subtraction modifies the target body by removing from it those pieces which overlap with the union of the tool bodies.

An intersection reduces the target body to those pieces which overlap with the union of the tool bodies.

Local booleans and trim with sheet, and booleans on general input bodies with acorn or wireframe components, are not yet supported.

Global Booleans: A global boolean is performed if both 'target' and 'tools' are bodies. One target body and one or more tool bodies are required. If more than one tool body is supplied, the union of overlapping tool bodies is computed first, and then the boolean between 'tools' and 'target' is performed.

Local Booleans: Local booleans can only be performed with SLIPGT set to zero.

A local boolean is performed if one or both of 'target' and 'tools' is a list of faces. 'Target' must contain one or more faces from a single body, and 'tools' must contain one or more faces from another body. Supplying a body for either 'target' or 'tools' (but not both), is equivalent to listing all the faces of the body.

In a local boolean, only the edges of intersection of the given faces are imprinted, unless the BOOPEF or BOOPSX options are selected (see below). The loops of imprinted edges divide the boundary of both bodies into facesets. For a global boolean all the facesets of the tool bodies lie completely inside or outside the target body. However, this is not necessarily the case for a local boolean, as not all faces of both bodies are used to compute the imprinted edges. For a local boolean, boundary facesets of the tool body are classified inside if they are locally inside the target body near a loop of imprinted edges, and outside if they are locally outside near the loop of imprinted edges.

The option BOOPEF may be used to ensure that the imprinted loops are complete.

The option BOOPSX may be used to eliminate the possibility of self intersecting results.

In a local boolean unite, boundary facesets of the tool body which are outside the target body are glued to the target. In a local boolean subtract, intersect or trim, only boundary facesets of the tool body inside the target body are used.

It is also possible for a local boolean to use a subset of these boundary facesets in computing the result. All facesets to be included in the result can be selected using the BOOPIC option, or all facesets to be excluded from the result can be selected using the BOOPEC option. It is not possible to select both BOOPIC and BOOPEC options, ie it is not possible to identify some facesets to be included, and some to be excluded, in the same operation. The facesets must be from the tool body. Each faceset must be unambiguously identified by a face, edge or vertex which is interior to it (see below).

For a local boolean there must be at least one intersection between the topology of 'target' and 'tools' (i.e. at least 1 face pair must clash). If this is not the case the boolean will be performed and the error KI_partial_no_intersect will be returned.

Merging and Tag Persistence: The result of a local or global boolean may contain a number of new mergable faces, edges and vertices. These are not merged away unless requested by selecting the BOOPME option. Only new mergable entities are deleted. The mergable faces, edges and vertices of the original target and tool bodies are not merged away.

Tag persistence rules for all global and local booleans can be summarised as follows:

- If a face will shrink (be truncated), then its tag will persist.
- If a face will grow as a result of merging two or more faces together, then the oldest target face will persist.
- If a face will be split into several faces, then one of the resulting faces will have the tag of the original face, and all other tags will be new.

Optimising Booleans: The boolean operation can be optimised, depending on the configuration of the target and tool bodies, when additional information is known regarding tool bodies not clashing with each other, or not clashing with edges on the 'target' face or body.

This optimisation is enabled by the options BOOCSH and BOOINF. These apply not only to instancing operations, but also to booleans with multiple tools which need not be instances of the same body.

If the BOOCSH option is used, the edges of each tool body may clash with the edges of the target body, but the tool bodies must not clash with each other.

If the BOOINF option is used, the tool bodies may clash with each other, but must not clash with existing edges on the target body.

A combination of both options will ensure no intersection tests are performed between the target and tool bodies.

The options which can be selected for both global and local booleans are summarised below:

Code in `opts'	Entry in `optdta'	Description
BOOPIN	NULTAG	Intersect 'target' with 'tools'. 'Target' is reduced to the vol ume (or area for sheet intersections) which overlaps 'tools'.
BOOPSU	NULTAG	Subtract 'tools' from 'target'. 'Target' is modified by removal of all volumes (or areas for sheet subtractions) which overlap 'tools'.
BOOPUN (default)	NULTAG	Unite 'tools' with 'target'. 'Target' is extended by the inclu sion of all volumes (or areas for sheet unions) contained in 'tools'. This is the default action if no code for type of bool ean is provided.
BOOPTS	NULTAG	Trim 'target' with sheet 'tools'. 'Target' is trimmed to all vol umes (or areas for sheet trims) behind the sheet 'tools'.

BOOPPU	NULTAG	Punch `target` sheet with solid `tools`. the result is the areas of the `target` sheet outside the solid `tools` olus the areas of the faces of the solid `tools' on one side of the sheet `target`. With BOOPUN this will be in front of the sheet and with BOOPSU it will be behind. This cannot be used with BOOPIN.
BOOPME	NULTAG	Merge all mergeable imprinted edges created by the boolean.
BOOCSH	NULTAG	None of the new 'tools' booleaned with the 'target' clash with each other. It may be the case that they clash with existing edges on the 'target', including the periphery edge.
BOOINF	NULTAG	None of the new 'tools' clash with any of the existing edges on the 'target', periphery or internal. It may be the case that the 'tools' clash with each other.

The options which can be selected for local booleans only are summarized as follows:

Code in `opts'	**Entry in `optdta'**	**Description**
BOOPEF	NULTAG	Extend face list 'target'. If the imprinting phase of the local boolean results in incomplete loops of imprinted edges, addi tional faces in 'target' will be used. This option has no effect if 'target' is a body.
BOOPSX	NULTAG	Stop self-intersections occuring in the result. The tool faces are compared with all faces of the target, to avoid self-inter sections in the result, outside the given faces of interset.
BOOPEC	face, edge or vertex, or list of same, one entity for each faceset to be excluded	Identify boundary facesets of the tool body to be excluded from the local boolean. All other useful boundary facesets will be used in the result. The list in 'optdta' must contain faces, edges or vertices of the tool body, which do not inter sect any of the 'target' faces.
BOOPIC	face, edge or vertex, or list of same, one entity for each faceset to be included	Identify boundary facesets of the tool body to be one included in the local boolean. All other boundary facesets will not be used in the result. The list in 'optdta' must contain faces, edges or vertices of the tool body, which do not inter sect any of the 'target' faces.

The options which are specific to global booleans when SLIPGT is set to one are summarised below:

Code in `opts'	Entry in `optdta'	Description
BOOPPS	NULTAG	Prune solid regions of the result. Any faces, edges, or vertices in the result body which are completely surrounded by solid regi will be deleted.
BOOPPV	NULTAG	Prune void regions of the result. Any faces, edges, or vertices in the result body which are completely surrounded by void regions will be deleted, so long as the result body contains some pieces of a higher dimension (for example, a sheet face will only be deleted if the result body contains at least one solid region, and so on).

CCLIST - Concatenate two lists, tail onto head

Receives

```
KI_tag_list          *head      list to be extended
KI_tag_list          *tail      list to be appended
```

Returns

```
KI_cod_error         *ifail     failure indicator
```

Specific Errors

```
KI_not_in_same_partition lists are not in the same partition
KI_bad_type_combn        head and tail lists are not of the same type
```

Description The 'tail' list is appended to the 'head' list. The 'tail' list is deleted. Both lists must be of the same type. If the tail list is owned before concatenation, it is detached prior to deletion. Ownership of the head list is unchanged.

Can be called from the GO.

CHCKEN - Checks an entity

Receives

```
KI_tag_list_entity      *entity         entity (or pair of entities) to
                                        check
<KI_int_nitems>         *mxflts         maximum number of faults to
                                        return
<KI_int_nitems>         *nopts          number of option codes
KI_cod_chop              option[nopts]  array of option codes
```

Returns

```
<KI_tag_list_int>       *faults         tokens describing faults in body
<KI_tag_list_<entity>>  *tags           faulty components
<KI_tag_list_<list>>    *pdata          list of data lists
<KI_int_nitems>         *nfault         number of faults returned
KI_cod_error            *ifail          failure indicator
```

Specific Errors

```
KI_unsuitable_entity    unsuitable entity
KI_bad_option_data      option inconsistent with type of entity
KI_list_wrong_length    wrong number of items in entity list
KI_duplicate_list_item  face appears twice in entity list
KI_not_in_same_body     faces are not both from the same body
```

Description This function performs a series of checks on an entity, or a list of two entities, and returns information about the faults found, if any. 'Entity' may be a body, face, edge, a list of two faces from the same body, or any geometric entity.

'nfault' gives the number of faults returned, and the lists 'faults', 'tags' and 'pdata' all contain 'nfault' elements. Information about each fault is returned in corresponding positions in the three lists.

If there are no faults found, 'nfault' is zero, and no other information is returned. Otherwise a token describing the fault is returned in 'faults', and for some types of fault there will also be a tag of the faulty entity in 'tags', and associated data in a real list whose tag is in 'pdata'. `pdata` is a list of tags which may be: the tag of a list of 3 reals representing a point; the tag of a list of tags (of edges or vertices) or the null tag.

If there is more than one fault in the entity then the function does not guarantee to return all the faults.

The caller may set an upper limit on the number of faults found, in 'mxflts'; if 'mxflts' is zero, no limit is applied.

The options allow the caller to control which checks are performed; if no options are supplied, all checks appropriate to the supplied entity will be made. The following table details the options available.

Token	Meaning	Applies to
CHOPCR	check for corrupt data structure, and check identifiers	bodies
CHOPIG	check for invalid, self-intersecting, or degenerate geometry	allentities
CHOPPV	check for invalid, self-intersecting, or degenerate geometry, ignore results of pre-V5 checks.	all entities
CHOPNO	check for invalid, self intersecting, or degenerate ge ometry, ignore previous checks and interface param eters.	all entities
CHOPED	check for inconsistencies between topology and geometry of edges	topological entities
CHOPFA	check for inconsistencies between topology and geometry of faces	bodies and faces
CHOPSX	check for self-intersecting faces	bodies and faces
CHOPLC	check for loop consistency of faces	bodies and faces
CHOPBX	check for size-box violations	topological entities

CHOPFF	check for face-face inconsistencies	bodies and pairs of faces
CHOPSH	check for inside-out or inconsistent shells	bodies

The option CHOPIG will check for invalid, self-intersecting and degenerate geometry, however the self-intersection checks are performed only if the appropriate interface parameter is set (see SEINTP). Geometry which passes theses tests is marked so that the cost of these tests need only be incurred once. Geometry which fails is also marked for the same reason.

The option CHOPPV behaves just like CHOPIG, except that it forces a re-check of geometry marked as valid by pre-V5 versions of Parasolid.

The option CHOPNO behaves just like CHOPIG, except that it always forces a full check of the geometry, overriding the interface parameter, even if the geometry has been marked as valid by a previous check.

When some types of check are omitted, an entity which would fail them may produce misleading results or checker failures if checks which appear later in the above table are applied. Very occasionally, checker failures (denoted by tokens RTSTCF and RTSTFC, see table below) are caused by difficult geometric configurations or the unexpected failure of an internal numerical algorithm; such failures should not be taken to indicate that the body is invalid.

The following table shows the tokens that may be returned, and the data associated with them. For more details on these fault types and how they may arise, see the chapter on 'Checking'.

Token	**Meaning**	**Associated tag**	**Associated data**
RTSTCR	data structure corrupt	null	null
RTSTID	invalid or duplicate identifiers	null	null
RTSTIG	invalid geometry	invalid geometric entity	null
RTSTGX	self-intersection in geometry	self-intersecting geomet ric entity	point on the self-intersection
RTSTDG	degenerate geometry	geometric entity	point from degenerate region
RTSTMG	missing geometry	topological entity with no geometry	null
RTSTOC	open or non-periodic curve on ring edge	edge	null
RTSTON	open or non-periodic nominal geometry on ring edge	edge	null
RTSTVC	vertex not on curve	edge	list of vertices
RTSTRN	vertex not on nominal geometry	edge	list of vertices
RTSTER	edge reversed	edge	null

RTSTRN	nominal geometry in wrong direction for edge	edge	null
RTSTSN	SP-curves of edge not within edge tolerance of nominal geometry	edge	null
RTSTSP	SP-curves of edge not within edge tolerance	edge	null
RTSTVT	vertices touch	edge	null
RTSTFO	faces incorrectly ordered at edge	edge	null
RTSTVS	vertex not on surface	face	list of vertices
RTSTES	edge not on surface	face	list of edges
RTSTEO	edges incorrectly ordered at vertex	face	null
RTSTMV	missing vertex at surface singularity	face	point at singularity
RTSTLC	loops inconsistent	face	null
RTSTSX	self-intersecting face i.e. edge/edge inconsistency	face	point in region of inconsistency
RTSTEF	wireframe edge/face incon sistency	list of tags of the edge and face	point in region of inconsistency
RTSTEE	wireframe edge/wireframe edge inconsistency	list of tags of the pair of edges	point in region of inconsistency
RTSTGC	geometry not G1 continu ous	face or edge	null
RTSTBX	size box violation	face or edge	null
RTSTFF	face-face inconsistency	list of tags of the pair of faces	point in region of inconsistency
RTSTNG	body is inside out	body	null
RTSTSH	shells of region are incon sistent	region	null
RTSTRS	regions of body are incon sistent	body	null
RTSTSG	geometry/topology incon sistency in shell	shell	null
RTSTAC	acorn shell/shell inconsis tency	list of tags of the pair of shells	point in region of inconsistency
RTSTCF	checker failure	topological entity	null
RTSTFC	failure during face-face check	list of tags of the pair of faces	null

Note: The tokens RTSTEF, RTSTFO, RTSTSG, RTSTAC can only occur for general bodies.

There are special checks performed on B-curves and B-surfaces.

The restrictions on B-curves are:

- A B-curve cannot have any first derivative of zero length.
- A B-curve cannot self-intersect.
- A B-curve cannot have a cusp interior to a segment.
- A B-curve must be at least position (G0) continuous.

The self intersection check in ii) is only performed if the appropriate interface parameter is set (see SEINTP).

The restrictions on B-surfaces are:

- A B-surface cannot contain any first derivative of zero length, except at a legal degeneracy (defined below).
- A B-surface cannot self-intersect.
- A B-surface cannot have a ridge or a cusp interior to a patch.
- A B-surface must be at least position (G0) continuous.
- A B-surface may meet itself in either the u or the v direction or both to form a closed surface. If it does meet itself, it must do so all along the boundary. It is illegal for for the surface to meet itself at a single point.
- A legal degeneracy is one which occurs only on a boundary of the surface, and which reduces all the boundary between two or more knot values to a single point.
- A corner of a B-surface cannot be degenerate in both parameters.

The self intersection check in ii) is only performed if the appropriate interface parameter is set (see SEINTP).

Offset surfaces whose underlying surfaces are B-surfaces are also checked for self intersection only if the appropriate interface parameter is set (see SEINTP).

SP-curves are also checked for self intersection only if the appropriate interface parameter is set (see SEINTP).

If `entity' is a body, face, or edge and option CHOPPA or CHOPED is selected, then additional checks on the geometry attached to `entity' are performed.

The checks performed on curves attached to edges are:

- The curve must be G1 continuous.
- If the curve is closed then it must have a periodic parametrisation.

The checks performed on surfaces attached to faces are:

- The surface must be G1 continuous.
- The constant parameter lines of the surface must be G1 continuous.
- If the surface is closed in either direction then it must have a periodic parametrisation in that direction.
- The surface may only contain degeneracies along one of its boundaries if all of that boundary is degenerate.

Failure to pass any of these checks will result in an RTSTGC code.

Checks i), iii), iv) and vi) are only performed if the appropriate interface parameter is set - see SEINTP (SLIPCO).

CLABYS - Clash bodies

Receives

```
KI_tag_body          *body1   first body
<KI_tag_transform>   *trans1  transformation for 'body1'
KI_tag_body          *body2   second body
<KI_tag_transform>   *trans2  transformation for 'body2'
KI_cod_logical       *lfull   true if exhaustive test required
```

Returns

```
KI_tag_list_face     *fclst1  list of faces in 'body1' which clash with
                              'body2'
KI_tag_list_face     *fclst2  list of faces in 'body2' which clash with
                              'body1'
<KI_int_nitems>      *nclash  length of face lists
KI_cod_error         *ifail   failure indicator
```

Specific Errors

```
KI_cant_do_clash  clash failure
KI_wire_body      one of the bodies is a wire
KI_wrong_transf   trans2 has shearing or non-uniform scaling; trans1
                  has shearing or non-uniform scaling
```

Description If 'lfull' is true, then 'body1', transformed by 'trans1', is compared with 'body2', transformed by 'trans2'. Every clash between a face of 'body1' and a face of 'body2' is entered in the lists, the face from 'body1' going into 'fclst1', and the face from 'body2' going into 'fclst2'. 'nclash' is set to the number of such clashes. A face may occur many times in one of the lists, paired with different faces from the other body. Two face lists will be returned even if 'nclash' is 1.

If 'lfull' is false, the test will proceed only until the first clash is found. If a clash is found, it will return the clashing faces (NOT lists) in 'fclst1' and 'fclst2', and return 'nclash' as 1.

If no clashes are found, irrespective of 'lfull', 'nclash' will be returned as 0 and 'fclst1' and 'fclst2' as the null tag.

If either of the transformations is given as the null tag, it will be taken to mean the identity transformation.

Only clashes between faces are found, therefore it is possible that no clashes will be found when in fact one body is completely contained in the other. Also, if either body is a general body, clashes between a wireframe edge in one body and a face or wireframe edge in the other will not be found.

If either of the transformations is supplied it may only contain translation, rotation, reflection and global scale components. Local scales and shears are not allowed.

CLENEN - Finds the closest point between two entities/ entity lists

Receives

```
KI_tag_list_entity    *ents1        1st entity/list of entities
```

```
KI_tag_list_entity    *ents2          2nd entity/list of entities
<KI_int_nitems>       *nopts          number of options
KI_cod_clop            iopts[nopts]   options
KI_tag_list_dbl        optdta[nopts]  option data
```

Returns

```
<KI_dbl_distance>     *mdist          minimum distance
KI_tag_list_<entity>  *elist1         entity list
KI_tag_list_<entity>  *elist2         entity list
KI_vec_position        entpt1         point on 1st entity
KI_vec_position        entpt2         point on 2nd entity
<KI_dbl>               parms1[2]      parameter/s related to entpt1
<KI_dbl>               parms2[2]      parameter/s related to entpt2
KI_tag_list_int       *props1         properties ( 1st entity)
<KI_int_nitems>       *nprop1         no of properties in props1
KI_tag_list_int       *props2         properties (2nd entity)
<KI_int_nitems>       *nprop2         no of properties in props2
KI_cod_error          *ifail          failure indicator
```

Specific Errors

```
KI_missing_geom             Topology without geometry supplied
KI_wrong_entity_in_list     Unsupported entity type supplied in list
KI_wrong_entity             Unsupported entity type supplied
KI_bad_option_data          Bad option data supplied
KI_closest_approach_failed  Failed to find closest approach
```

Description CLENEN will find the closest approach between 'ents1' and ents2', where 'ents1' and 'ents2' can either be single entities or lists of entities.

The following entity types are supported:

Geometrical	point, curve and surface
Topological	vertex, edge, face and body

Curves and surfaces must be legally attachable to edges and faces, and must be G1 continuous.

The entities supplied must either all be geometry or all be topology.

The minimum distance ('mdist') will be returned together with a single pair of solution points. The pair of solution points will be returned as two vectors ('entpt1' and 'entpt2'), the minimum distance being achieved between these two vectors. Only one pair of solution points will be returned, regardless of how many possible solution point pairs there are.

The entities and sub-topology upon which solution points lie will be identified within the respective entity lists 'elist1' and 'elist2'. These entity lists will be of length 2:

- The first list component will be the tag of the (received) entity upon which the solution point was found to lie.
- The second list component will indicate a sub-topology upon which the solution point was found to lie. This sub-topology will be either a face, edge or vertex, belonging to a received entity. If the solution does not lie on sub-topology a 'NULTAG' will be returned.

If either of the entities in the list has a tolerance associated with it, the solution on that entity will only be accurate and unique up to the tolerance of the entity.

Solution Parametrisation: When appropriate curve or surface parameter/s will also be returned (in the arrays 'parms1' and 'parms2'). What parametrisation is used is firstly dependent upon the geometry of entity returned and secondly upon the geometry of the sub-topology (if identified). Both arrays will be of length 2, but if curve parametrisation is used the 2nd array component should be ignored.

The following table indicates what parametrisation is used and when:

Entity	**Identified Sub-topology**	**Geometry used for Solution Parametrisation**
point	none	none
curve	none	curve
surface	none	surface
vertex	none	none
edge	none	curve of the edge
edge	vertex	curve of the edge
face	none	surface of the face
face	edge	surface of the face
face	vertex	surface of the face
body	face	surface of the face
body	edge	curve of the edge
body	vertex	none

The curve and surface parameter values returned, will be the same as would be returned by the parametrisation functions ENPAPC, and ENPAPS.

Options: Various options are available through the use of 'nopts' 'iopts' & 'optdta'. 'nopts' specifies the number of options requested, 'iopts' is an array of option types, and 'optdta' is an array of lists of corresponding option data. Some options are only available for restricted cases.

The following 3 options are available regardless of how many entities or what entity types are supplied:

- An upper bound on the minimum distance to be computed may be supplied. If such a bound is supplied the minimum distance achieved will only be identified if it is less than the bound.
- A lower bound on the minimum distance. For this option, if the closest approach between an 'entity' and 'point' is found to be less than this bound, no solution will be

determined and a token will be returned to indicate that the minimum distance is less than this bound.

- A tolerance on the accuracy of the minimum distance measurement. This will allow slacker computation of minimum distance, when the default accuracy is not required. The default accuracy will be the linear precision of the modeler (set up by SEMODP).

The following additional options will be available but only for limited cases. These options must only be allowed to refer to a single entity, and not a list of entities nor a single component of an entity list.

- For a single curve or edge, a curve parameter estimate may be supplied.
- For a single surface or face, surface parameter estimates may be supplied.
- For a single curve, surface, edge, or face, a position vector estimate may be supplied. Such an estimate is expected to be on or close to the entity.

Both parameter and position vector estimates cannot be supplied for an entity.

The use of any of the above estimates is not allowed if the minimum distance computation involves a body (e.g. a face cannot be given estimates for a face-body computation).

The following table details the tokens and data required for the options.

Option	Token	Option Data (real values)
To supply an upper distance bound	CLOPUP	1 real: the upper bound
To supply a lower distance bound	CLOPLW	1 real: the lower bound
To supply a tolerance	CLOPTL	1 real: the tolerance
To supply a pvec estimate	CLOPPT	4reals: The 1st to indicate whether the data applies to the 1st or second entity argu ment (must equal 1.0 or 2.0) The last 3 to indicate the vector
To supply a parameter estimate for the 1st entity	CLOPP1	1 or 2 reals: 1 curve or 2 surface parameter estimates
To supply a parameter estimate for the 2nd entity	CLOPP2	1 or 2 reals: 1 curve or 2 surface parameter estimates

Properties: A list of tokens, 'props' is returned to indicate the following:

- The solution as being a regional solution upon the entity (i.e it is possible to move from solution point pair without the distance changing) : RTCLRS

The identification of such a regional solution is not guaranteed.

- No solution being returned because the distance was found to be less than the lower bound : RTCLLB
- No solution being returned because no distance was found to be less than the upper bound : RTCLUB
- For a body only, the distance as being negative. The distance will be classified as negative if the solution point on the other entity lies inside the body : RTCLND

CLENEX - Finds the closest point between two entities/ entity lists.

Receives

```
KI_tag_list_entity   *ents1         1st entity/list of entities
KI_tag_list_entity   *ents2         2nd entity/list of entities
<KI_int_nitems>      *nopts         number of options
KI_cod_clop          iopts[nopts]   options
<KI_tag_list_dbl>    optdta[nopts]  option data
```

Returns

```
<KI_int_nitems>      *nmins         number of minima returned
<KI_tag_list_dbl>    *min_dists     distances
<KI_tag_list_<list>> *elists1       lists of entities
<KI_tag_list_<list>> *elists2       lists of entities
<KI_tag_list_list>   *entpts1       point on 1st entity
<KI_tag_list_list>   *entpts2       point on 2nd entity
<KI_tag_list_list>   *parms1        parameter/s related to entpt1
<KI_tag_list_list>   *parms2        parameter/s related to entpt2
<KI_tag_list_<list>> *props1        properties ( 1st entity)
<KI_tag_list_int>    *nprops1       no of properties in props1
<KI_tag_list_<list>> *props2        properties (2nd entity)
<KI_tag_list_int>    *nprops2       no of properties in props2
KI_cod_error         *ifail         failure indicator
```

Specific Errors

```
KI_missing_geom             Topology without geometry supplied
KI_wrong_entity_in_list     Unsupported entity type supplied in list
KI_wrong_entity             Unsupported entity type supplied
KI_bad_option_data          bad option data supplied
KI_closest_approach_failed  failed to find closest approach
```

Description CLENEX will find global or local closest approaches between 'ents1' and ents2'. Where a single global closest approach is requested 'ents1' and 'ents2' can either be single entities or lists of entities. If all local closest approaches are required then both 'ents1' and 'ents2' must be single entities.

The following entity types are supported:

Geometrical	point, curve and surface
Topological	vertex, edge, face and body

Curves and surfaces must be legally attachable to edges and faces, and must be G1 continuous.

The entities supplied must either all be geometry or all be topology.

The function returns the number of closest approaches found and collection of lists all of the same length. Taken together the n'th entries of these list form a description of the n'th closest approach found, as follows:

The entry from the list 'min_dists' is a real value for the distance of the closest approach, and 'entpts1' and 'entpts2' corresponding entries hold the points between which the closest approach is achieved (as lists of 3 reals). The entries in 'elists1' and 'elists2' are lists, of length 2, which identify the entities and sub-topology upon which those solution points lie. These lists will be structured as follows:

- The first list component will be the tag of the (received) entity upon which the solution point was found to lie.
- The second list component will indicate a sub-topology upon which the solution point was found to lie. This sub-topology will be either a face, edge or vertex, belonging to a received entity. If the solution does not lie on sub-topology a 'NULTAG' will be returned.

If either of the entities in the list has a tolerance associated with it, the solution on that entity will only be accurate and unique up to the tolerance of the entity.

Solution parametrisation is provided by the entries in the 'parms1' and 'parms2' lists when appropriate. These entries are real lists of length 2 holding curve or surface parameter/s. Which parametrisation is used is firstly dependent upon the geometry of entity returned and secondly upon the geometry of the sub-topology (if identified). If curve parametrisation is used the 2nd list component should be ignored.

The table below indicates what parametrisation is used and when:

Entity	**Identified Sub-topology**	**Geometry used for Solution Parametrisation**
point	none	none
curve	none	curve
surface	none	surface
vertex	none	none
edge	none	curve of the edge
edge	vertex	curve of the edge
face	none	surface of the face
face	edge	surface of the face
face	vertex	surface of the face
body	face	surface of the face

body	edge	curve of the edge
body	vertex	none

The curve and surface parameter values returned, will be the same as would be returned by the parametrisation functions ENPAPC, and ENPAPS.

Properties: Properties of the closest approach are returned through the entries in the 'props1' and 'props2' lists. Each entry is a list of tokens (of variable lengths given by the entries in 'nprops1' and 'nprops2'). These properties and their tokens are as follows:

- RTCLRS: The solution is a regional solution upon the entity (i.e it is possible to move from solution point pair without the distance changing The identification of such a regional solution is not guaranteed.
- RTCLND: For a body only, the distance as being negative. The distance will be classified as negative if the solution point on the other entity lies inside the body.

In the case that there are no solutions, i.e. upper or lower bounds (see below) have been set that rule out all closest approaches, 'nmins' will be set to 0 and the lists 'props1' and 'props' will be of length 1. The only entry in these lists will be a list holding a single token indicating why no solutions could be found:

- RTCLLB No solution being returned because the distance was found to be less than the lower bound.
- RTCLUB: No solution being returned because no distance was found to be less than the upper bound.

Options: Various options are available through the use of 'nopts' 'iopts' & 'optdta'. 'nopts' specifies the number of options requested, 'iopts' is an array of option types, and 'optdta' is an array of lists of corresponding option data. Some options are only available for restricted cases.

The following option, 'CLOPTL', is available under all circumstances:

- A tolerance on the accuracy of the minimum distance measurement. This will allow slacker computation of minimum distance, when the default accuracy is not required. The default accuracy will be the linear precision of the modeller (set up by SEMODP).

An option, 'CLOPFA', is available to request that all local closest approaches should be returned. At present this option is incompatible with all the other options mentioned below. It is also only available in the case that 'ents1' and 'ents2' are single entities.

If the option for searching for all closest approaches is omitted, then the following options are available. Some of these do have restrictions of their own.

The following options are available regardless of how many entities or what entity types are supplied:

- An upper bound on the minimum distance to be computed may be supplied. If such a bound is supplied the minimum distance achieved will only be identified if it is less than the bound.
- A lower bound on the minimum distance. For this option, if the closest approach between entities is found to be less than this bound, no solution will be determined and a token will be returned to indicate that the minimum distance is less than this bound.

The following additional options will be available but only for limited cases. These options must only be allowed to refer to a single entity, and not a list of entities nor a single component of an entity list.

- For a single curve or edge, a curve parameter estimate may be supplied.
- For a single surface or face, surface parameter estimates may be supplied.
- For a single curve, surface, edge, or face, a position vector estimate may be supplied. Such an estimate is expected to be on or close to the entity.

Both parameter and position vector estimates cannot be supplied for an entity.

The use of any of the above estimates is not allowed if the minimum distance computation involves a body (e.g. a face cannot be given estimates for a face-body computation).

The table below details the tokens and data required for the options.

Option	**Token**	**Option Data (real values)**
To supply an upper distance bound	CLOPUP	1 real: the upper bound
To supply a lower distance bound	CLOPLW	1 real: the lower bound
To supply a tolerance	CLOPTL	1 real: the tolerance
To supply a pvec estimate	CLOPPT	4reals: The 1st to indicate whether the data applies to the 1st or second entity argu ment (must equal 1.0 or 2.0) The last 3 to indicate the vector
To supply a parameter estimate for the 1st entity	CLOPP1	1 or 2 reals: 1 curve or 2 surface parameter estimates
To supply a parameter estimate for the 2nd entity	CLOPP2	1 or 2 reals: 1 curve or 2 surface parameter estimates
To request that all local closest approaches be returned	CLOPFA	No data required

CLPTEN - Finds the closest point on an entity to a given point

Receives

```
KI_vec_position          point           point
KI_tag_list_entity      *ents            entities
<KI_int_nitems>         *nopts           number of options
KI_cod_clop              iopts[nopts]    options
KI_tag_list_dbl          optdta[nopts]   option data
```

Returns

```
<KI_dbl_distance>       *mdist           minimum distance
KI_tag_list_<entity>    *elist           entity list
KI_vec_position          epoint          point on entity
```

```
<KI_dbl>              csparm[2]    curve or surface parameters
KI_tag_list_int      *props        properties
<KI_int_nitems>      *nprops       no. of properties
KI_cod_error         *ifail        failure indicator
```

Specific Errors

```
KI_missing_geom              Topology without geometry supplied
KI_wrong_entity_in_list      Unsupported entity type supplied in list
KI_wrong_entity              Unsupported entity type supplied
KI_bad_option_data           Bad option data supplied
KI_closest_approach_failed   Failed to find closest approach
```

Description Given an entity or list of entities ('ents'), CLPTEN will determine an entity point ('epoint') which is the minimum distance ('mdist') from the received 'point'. Only one 'epoint' will be returned, regardless of how many possible points achieve the minimum distance.

The following entity types are supported:

Geometrical	point, curve and surface
Topological	vertex, edge, face and body

Curves and surfaces must be legally attachable to edges and faces, and must be G1 continuous.

The returned entity list ('elist') will contain two components:

- The first list component will be the tag of the (received) entity upon which the solution was found to lie.
- The second list component will indicate a sub-topology upon which the solution was found to lie. This sub-topology will be either a face, edge or vertex, belonging to a received entity. If the solution does not lie on sub-topology a 'NULTAG' will be returned.

If either of the entities in the list has a tolerance associated with it, the solution on that entity will only be accurate and unique up to the tolerance of the entity.

Solution Parametrisation:

When appropriate a curve parameter or 2 surface parameters will be returned in the 'csparm' array. This array is of length 2, but for the curve parameter case the 2nd array entry should be ignored. What parametrisation is used is firstly dependent upon the geometry of the entity returned and secondly upon the geometry of the sub-topology (if identified). The table below indicates what parametrisation is used and when:

Entity	**Identified Sub-topology**	**Geometry used for Solution Parametrisation**
point	none	none
curve	none	curve
surface	none	surface
vertex	none	none

edge	none	curve of the edge
edge	vertex	curve of the edge
face	none	surface of the face
face	edge	surface of the face
face	vertex	surface of the face
body	face	surface of the face
body	edge	curve of the edge
body	vertex	none

The curve and surface parameter values returned, will be the same as would be returned by the parametrisation functions ENPAPC, and ENPAPS.

Options: Various options are available through the use of 'nopts' 'iopts' & 'optdta'. 'nopts' specifies the number of options requested, 'iopts' is an array of option types, and 'optdta' is an array of lists of corresponding option data. Some options are only available for restricted cases.

The following 3 options are available regardless of how many entities or what entity types are supplied:

- An upper bound on the minimum distance to be computed may be supplied. If such a bound is supplied the minimum distance achieved will only be identified if it is less than the bound.
- A lower bound on the minimum distance. For this option, if the closest approach between an 'entity' and 'point' is found to be less than this bound, no solution will be determined and a token will be returned to indicate that the minimum distance is less than this bound.
- A tolerance on the accuracy of the minimum distance measurement. This will allow slacker computation of minimum distance, when the default accuracy is not required. The default accuracy will be the linear precision of the modeler (set up by SEMODP). If supplied the tolerance must not be less than the default value.

If both lower and upper bounds are supplied it is illegal for the lower bound to be greater than the upper bound.

The following additional options will be available but only for limited cases. These options must only be allowed to refer to a single entity, and not a list of entities nor a single component of an entity list.

- For a single curve or edge, a curve parameter estimate may be supplied.
- For a single surface or face, surface parameter estimates may be supplied.
- For a single curve, surface, edge, or face, a position vector estimate may be supplied. Such an estimate is expected to be on or close to the entity.

Both parameter and position vector estimates cannot be supplied for an entity.

The table below details the tokens and data required for the options.

Option	Token	Option Data (real values)
To supply an upper distance bound	CLOPUP	1 real: the upper bound
To supply a lower distance bound	CLOPLW	1 real: the lower bound
To supply a tolerance	CLOPTL	1 real: the tolerance
To supply a pvec estimate	CLOPPT	3 reals: vector estimate
To supply a parameter estimate	CLOPP1	1 or 2 reals: 1 curve or 2 surface parameter

Properties: A list of tokens, 'props' is returned to indicate the following:

- The solution as being a regional solution upon the entity (i.e it is possible to move along the entity from the solution point without the distance changing) : RTCLRS
- The identification of such a regional solution is not guaranteed.
- No solution being returned because the distance was found to be less than the lower bound : RTCLLB
- No solution being returned because no distance was found to be less than the upper bound : RTCLUB
- For a body only, the distance as being negative. The distance will be classified as negative if the 'point' lies inside the body : RTCLND

CLPTEX - Finds the closest point on an entity to a given point.

Receives

```
KI_vec_position       point          point
KI_tag_list_entity   *ents           entity/list of entities
<KI_int_nitems>      *nopts          number of options
KI_cod_clop           iopts[nopts]   options
<KI_tag_list_dbl>     optdta[nopts]  option data
```

Returns

```
<KI_int_nitems>      *nmins          number of minima returned
<KI_tag_list_dbl>    *min_dists      distances
<KI_tag_list_<list>> *elists         lists of entities
<KI_tag_list_list>   *entpts         point on entity
<KI_tag_list_list>   *parms          parameter/s related to entity
<KI_tag_list_<list>> *props          properties ( 1st entity)
<KI_tag_list_int>    *nprops         no of properties in props
KI_cod_error         *ifail          failure indicator
```

Specific Errors

```
KI_missing_geom             Topology without geometry supplied
KI_wrong_entity_in_list     Unsupported entity type supplied in list
KI_wrong_entity             Unsupported entity type supplied
KI_bad_option_data          bad option data supplied
KI_closest_approach_failed  failed to find closest approach
```

Description CLPTEX will find global or local closest approaches between a 'point' and 'ents'. Where a single global closest approach is requested 'ents' can either be a single entity or list of entities. If all local closest approaches are required then 'ents' must be a single entity.

The following entity types are supported:

Geometrical	point, curve and surface
Topological	vertex, edge, face and body

Curves and surfaces must be legally attachable to edges and faces, and must be G1 continuous.

The entities supplied must either all be geometry or all be topology.

The function returns the number of closest approaches found and a collection of lists all of the same length. Taken together the n'th entries of these lists form a description of the n'th closest approach found, as follows:

The entry from the list 'min_dists' is a real value for the distance of the closest approach, and 'entpts' corresponding entry holds the point at which the closest approach is achieved (as lists of 3 reals). The entry in 'elists' is a list, of length 2, which identifies the entity and sub-topology upon which the solution point lies. This list will be structured as follows:

- The first list component will be the tag of the (received) entity upon which the solution point was found to lie.
- The second list component will indicate a sub-topology upon which the solution point was found to lie. This sub-topology will be either a face, edge or vertex, belonging to a received entity. If the solution does not lie on sub-topology a 'NULTAG' will be returned.

If either of the entities in the list has a tolerance associated with it, the solution on that entity will only be accurate and unique up to the tolerance of the entity.

Solution parametrisation is provided by the entries in the 'parms' list when appropriate. These entries are real lists of length 2 holding curve or surface parameter/s. What parametrisation is used is firstly dependent upon the geometry of entity returned and secondly upon the geometry of the sub-topology (if identified). If curve parametrisation is used the 2nd list component should be ignored.

The table below indicates what parametrisation is used and when:

Entity	**Identified Sub-topology**	**Geometry used for Solution Parametrisation**
point	none	none
curve	none	curve
surface	none	surface
vertex	none	none
edge	none	curve of the edge
edge	vertex	curve of the edge

face	none	surface of the face
face	edge	surface of the face
face	vertex	surface of the face
body	face	surface of the face
body	edge	curve of the edge
body	vertex	none

The curve and surface parameter values returned, will be the same as would be returned by the parametrisation functions ENPAPC, and ENPAPS.

Properties: Properties of the closest approach are returned through the entries in the 'props' list. Each entry is a list of tokens (of variable length given by the entry in 'nprops'). These properties and their tokens are as follows:

- RTCLRS: The solution is a regional solution upon the entity (i.e it is possible to move from solution point pair without the distance changing) The identification of such a regional solution is not guaranteed.
- RTCLND: For a body only, the distance as being negative. The distance will be classified as negative if the solution point on the other entity lies inside the body

In the case that there are no solutions, i.e. upper or lower bounds have been set (see below) that rule out all closest approaches, 'nmins' will be set to 0 and the list 'props' will be of length 1. The only entry in this list will be a list holding a single token indicating why no solutions could be found:

- RTCLLB No solution being returned because the distance was found to be less than the lower bound
- RTCLUB: No solution being returned because no distance was found to be less than the upper bound

Options: Various options are available through the use of 'nopts' 'iopts' & 'optdta'. 'nopts' specifies the number of options requested, 'iopts' is an array of option types, and 'optdta' is an array of lists of corresponding option data. Some options are only available for restricted cases.

The following option, 'CLOPTL', is available under all circumstances:

- A tolerance on the accuracy of the minimum distance measurement. This will allow slacker computation of minimum distance, when the default accuracy is not required. The default accuracy will be the linear precision of the modeller (set up by SEMODP).

An option, 'CLOPFA', is available to request that all local closest approaches should be returned. At present this option is incompatible with all the other options mentioned below. It is also only available in the case that 'ents' is a single entity

If the option for searching for all closest approaches is omitted, then the following options are available. Some of these do have restrictions of their own.

The following options are available regardless of how many entities or what entity types are supplied:

- An upper bound on the minimum distance to be computed may be supplied. If such a bound is supplied the minimum distance achieved will only be identified if it is less than the bound.
- A lower bound on the minimum distance. For this option, if the closest approach between an entity and 'point' is found to be less than this bound, no solution will be determined and a token will be returned to indicate that the minimum distance is less than this bound.

The following additional options will be available but only for limited cases. These options must only be allowed to refer to a single entity, and not a list of entities nor a single component of an entity list.

- For a single curve or edge, a curve parameter estimate may be supplied.
- For a single surface or face, surface parameter estimates may be supplied.
- For a single curve, surface, edge, or face, a position vector estimate may be supplied. Such an estimate is expected to be on or close to the entity.

Both parameter and position vector estimates cannot be supplied for an entity.

The use of any of the above estimates is not allowed if the minimum distance computation involves a body (e.g. a face cannot be given estimates for a face-body computation).

The table below details the tokens and data required for the options.

Option	**Token**	**Option Data (real values)**
To supply an upper distance bound	CLOPUP	1 real: the upper bound
To supply a lower distance bound	CLOPLW	1 real: the lower bound
To supply a tolerance	CLOPTL	1 real: the tolerance
To supply a pvec estimate	CLOPPT	3 reals: vector estimate
To supply a parameter estimate	CLOPP1	1 or 2 reals: 1 curve or 2 surface parameter estimates
To request that all local closest approaches be returned	CLOPFA	No data required

CLPTFA - Finds closest point on a face to a given point

Receives

```
KI_vec_position    point          point
KI_tag_face       *face           face
<KI_int_nitems>   *nopts          number of options
KI_cod_clop        iopts[nopts]   estimation options
KI_tag_list        optdta[nopts]  option data
```

Returns

```
KI_vec_position    fpoint         point on face
```

```
KI_dbl_parameter   params[2]      surface parameters at 'point'
<KI_tag_entity>   *topol          topology at closest point
KI_cod_logical    *ortho          indicates whether soln orthogonal
KI_cod_error      *ifail          failure indicator
```

Specific Errors

```
KI_bad_parameter            parameter out of range
KI_closest_approach_failed  failed to find closest approach
KI_bad_option_data          bad option data
KI_missing_geom             supplied face has no associated surface
```

Description Returns the point 'fpoint' on the face 'face' which is closest to the point 'point'. If the closest point lies on the boundary of the face the corresponding edge or vertex is returned in 'topol', otherwise for a solution inside the boundary of the face NULTAG is returned.

If the entity returned in 'topol' has a tolerance associated with it, the solution 'fpoint' will only be accurate and unique to the tolerance of that entity.

If logical code 'ortho' equals KI_true, either the points 'point' and 'fpoint' are coincident or the vector between them is orthogonal to the face. In other words, when 'ortho' = KI_true the point enquired from ('point') can be obtained by offsetting from the solution point ('fpoint') along the surface normal (in the +/- direction). 'ortho' = KI_false should only be expected if the solution lies on an edge or vertex.

This function cannot always be guaranteed to find an appropriate point on the face, it will perform best if the supplied point is relatively close to the face. A solution will only be returned if it is unique.

The chance of success is greatly increased if an estimate of the closest point on the face is supplied. Estimates may be supplied as a point known to be close to the solution, or by giving the parameters of such a point specified in the parameter space of the surface. The form of the surface parametrisations are given in the documentation for ENSUPA. In general optimal performance is obtained if both a point and its parameters are supplied.

Estimate types are supplied in the array 'iopts' and the corresponding estimate is supplied in 'optdta'. The permissible entries in these arrays are:

`iopts' entry	`optdta' entry	Description
CLOPPT	list of 3 doubles	Specify coordinates of point close to solution
CLOPPR	list of 2 doubles	Specify parameters corresponding to a point close to the solu tion

COFEAT - Count entities in feature

Receives

```
KI_tag_feature      *featre    feature
```

Returns

```
<KI_int_nitems>     *nitems    number of items in feature
KI_cod_error        *ifail     failure code
```

Description The number of entities in the feature is returned in 'nitems'.

Can be called from the GO.

COLIST - Count items in a list

Receives

```
KI_tag_list            *list     list in which to count items
```

Returns

```
<KI_int_nitems>        *nitems   number of items in list
KI_cod_error           *ifail    failure indicator
```

Description The length of the given list is returned in 'nitems'.

Can be called from the GO.

COMENT - Comment the journal file

Receives

```
<KI_int_nitems>     *nchars          number of chars in 'coment'
KI_chr_string        coment[nchars]  comment string
```

Returns

```
KI_cod_error        *ifail           failure code
```

Description If journalling is on then 'coment' is journalled as a comment.

Can be called from the GO.

COPYEN - Copy entity

Receives

```
KI_tag_entity         *oldent      entity to be copied
```

Returns

```
KI_tag_entity         *newent      copy of entity
KI_cod_error          *ifail       failure indicator
```

Specific Errors

```
KI_wrong_entity      can't copy entity of given type
KI_part_not_isolated copy would instance true-sub-part of stored part
```

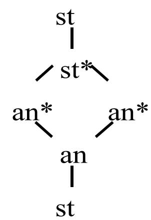
Description COPYEN returns a copy of 'oldent' in 'newent'. The types of entity which may be copied are:

- Assembly: - 'newent' will be an uninstanced copy of 'oldent'. The instances in 'newent' will be copies of those in 'oldent' but the parts they reference will not be copied. Any features and attributes attached to the assembly are also copied. The state of 'newent' will be new (ENSTNW).

  If 'oldent' is an unloaded (ENSTUN) assembly it is received before copying is attempted.

  We define the true-sub-parts of a stored (ENSTST) part S as those anonymous (ENSTAN) sub-parts of S reachable from S without encountering other stored parts.

  A stored or anonymous assembly which instances true-sub-parts of a stored part may not be copied as this would cause the stored part to become unisolated. For example, the parts marked with a * may not be copied and KI_part_not_isolated will be returned in 'ifail'.



- Instance: - If 'oldent' is an instance of part P in assembly A then 'newent' will be a new instance of P in A with a copy of 'oldent's transformation. Any attributes attached to the instance are not copied.
- Body: - 'newent' will be an uninstanced copy of 'oldent'. Any features and attributes attached to the body will also be copied. The state of 'newent' will be new (ENSTNW).

  If 'oldent' is an unloaded (ENSTUN) body it is received before copying is attempted.
- Surface: -
- Curve: -
- Point: - 'newent' will be an copy of 'oldent' plus any underlying geometry. If 'oldent' is construction geometry in a part 'newent' will also be construction geometry in the same part. Otherwise 'newent' will be orphaned. Any attributes attached to 'oldent' will not be copied.
- Transformation: - 'newent' is an orphaned copy of 'oldent'.
- List: - 'newent' is an orphaned copy of 'oldent'. In the case of a tag list the entities whose tags are in the list are not copied.
- Feature: - 'newent' will have the same members as 'oldent' and will be in the same part. Any attributes attached to 'oldent' will not be copied.

  If 'oldent' is not one of these types KI_wrong_entity is returned in 'ifail'.

CRATDF - Create a new attribute type definition

Receives

KI_int_nitems	*namlen	length of 'name'
KI_chr_string	name[namlen]	name of attribute type
KI_int_nitems	*nopts	number of option codes
KI_cod_atop	option[nopts]	array of option codes
<KI_tag_list>	opdata[nopts]	corresponding lists of data

Returns

KI_tag_attrib_def	*type	attribute type
KI_cod_error	*ifail	error code

Specific Errors

KI_illegal_owner	invalid class for type with geometric owner
KI_not_found	legal owner option has been omitted; class code has been omitted; no legal owners supplied
KI_bad_request_code	invalid class code supplied; list of field types contains an invalid code
KI_list_wrong_length	class code list length is not one
KI_not_unique	a token has been repeated in 'option'

```
KI_wrong_entity_in_list  invalid token found in list of legal owners
KI_duplicate_list_item   item duplicated in list of legal owners
KI_existing_attr_type    attribute type already defined
KI_bad_name              invalid name for attribute type
```

Description Creates a new attribute type definition with the given name, returning the tag of the type for use in calling the other attribute functions.

The attribute type definition comes in two parts; the name, which will act as a label for the type which is fixed across transmit and receive, and the option data, which specify what entities may own attributes of the given type, what fields these attributes have and how such attributes behave when modelling operations are applied to their owners.

Type name: - The name is specified by a character array containing only printable characters and an integer argument stating the length of the name.

If 'name' begins with the string SDL/TY, which is reserved for attribute types whose names are generated internally by Parasolid, KI_bad_name will be returned in 'ifail'.

Option data: - The data defining the attribute type are supplied in the arrays 'option' and 'opdata'. Each entry in 'option' is a code from the sequence ATOP00. The entries in 'opdata' are tags of lists of data appropriate to the codes in the corresponding positions in 'option'.

The following table lists the legal 'option' values and the corresponding data; the first two codes must be supplied to specify the legal owners and behavior when modelling operations are applied to the owner of an attribute:

Token Option [1]	**data in List [1]**
ATOPCL	RQAC (class) code indicating behaviour when owner changes
ATOPOW	legal owner type codes
ATOPFL	field types

Field types: - Every attribute of a certain type contains a specified sequence of fields of specified types; these are supplied to CRATDF as request codes in an integer list in the position in array 'opdata' corresponding to that of the entry ATOPFL in the array 'option'. The permitted tokens and their meanings are as follows:

Token	**Meaning**
RQAPRL	Real
RQAPIN	Integer
RQAPCS	Character string
RQAPVC	Vector
RQAPCO	Co-ordinate
RQAPDR	Direction
RQAPAX	Axis

An axis field consists of two vector values, communicated as six real values, one of which is a point on the axis and the other is its direction.

The order of the fields is important, as they are distinguished in OUATDF, CREATT and OUTATT by type and order.

To define an attribute which has no fields, omit ATOPFL in 'option'.

Class code: - The behavior of an attribute of the new type when the entity to which the attribute is attached is changed by a modelling operation is described by the request code supplied in 'opdata' alongside the code ATOPCL in 'option'. The meaning of the request code alongside ATOPCL is as follows:

Code	Meaning
RQAC01	Attribute is independent of physical size and position of entity to which it is at tached (e.g. density).
RQAC02	Attribute is dependent on entity size but not on position, (e.g. weight).
RQAC03	Attribute may vary with position or orientation (e.g. moment of inertia).
RQAC04	Attribute transforms with its owner, but is otherwise independent of the size and shape of its owner (e.g. start-point or direction of movement of the tool that cuts a face)
RQAC05	Attribute transforms with its owner provided its owner is not changed in other ways (e.g. center of gravity)
RQAC06	As for class 1, attribute is independent of physical size and position of the entity to which it is attached. However this class of attribute supports multiple values - one entity may have a list of attributes of the same type attached.
RQAC07	As for class 4, attribute transforms with its owner, but is otherwise independent of size and shape of the owner. However this class supports multiple values - an enti ty may have a list of attributes of the same type attached.

The response of a field of an attribute when the attribute transforms (with its owner) depends on the type of the field as follows:

- real, integer and string fields are unaffected,
- co-ordinate fields are acted on by the transformation,
- vector and direction fields are acted on by the reflection and rotation parts of the transformation, and
- axis fields behave as a coordinate field and a direction field.

For a fuller account of the behavior of attributes under modelling operations and the meaning of attributes, consult the chapter on Attributes.

Legal owners: - The entities which may legitimately own attributes of the given type are specified in an integer list, passed in 'opdata' in the position corresponding to that of the code ATOPOW in 'option', using the following token values to identify the different entity types:

Token	Entity Type
TYTOAS	Assembly
TYTOIN	Instance
TYTOBY	Body

TYTORG	Region
TYTOSH	Shell
TYTOFA	Face
TYTOLO	Loop
TYTOED	Edge
TYTOVX	Vertex
TYADFE	Feature
TYGESU	Surface
TYDECU	Curve
TYGEPT	Point

The geometric entity types, surface, curve and point, may only be specified as legal owners for attribute types with class RQAC01 or RQAC04.

CRBSPC - Create B-curve from B-spline data

Receives

```
KI_int_dimension     *dim              dimension of control points
KI_int_order         *order            order of curve
KI_int_nitems        *nctrl            number of control points
KI_dbl_coefficients   ctrl[dim*nctrl]  control points
KI_dbl_knots          knots[]          knot vector
<KI_int_nitems>      *nprops           number of curve properties
KI_cod_papr           props[nprops]    array of curve properties
```

Returns

```
KI_tag_b_curve       *bc               B-curve
KI_cod_error         *ifail            failure indicator
```

Specific Errors

```
KI_bad_knots                    invalid knot vector
KI_weight_le_0                  weights are non positive
KI_insufficient_points          insufficient control points
KI_bad_parametric_prop          property not applicable
```

Description This function creates a B-curve from B-spline data - i.e. a set of B-spline vertices (control points), and a knot vector. The curve may be either periodic or non-periodic, and the 'props' array supplies this information.

Dimension of control points 'dim':

- For rational curves 'dim'=4.
- For non-rational curves 'dim'=3.

Order of the curve 'order':

- The order of the curve = degree + 1.
- The minimum order is 2.

Number of control points 'nctrl':

- For non-periodic curves 'nctrl' >= 'order'.
- For periodic curves 'nctrl' >= 3.

Control points 'ctrl':

- For non-rational curves, the control points are points in 3-space and must be supplied [x0,y0,z0,x1,y1,z1,...].
- For rational curves each vector contains a point in 3-space followed by a weight for the point. The points are supplied [x0,y0,z0,w0,x1,y1,z1,w1,...]. The weight must be positive. Increasing the weight of a control point will pull the curve closer to that point.

Knot vector 'knots':

- The knot values must form a non-decreasing sequence.
- For non-periodic curves there must be ('nctrl' + 'order') knot values, the maximum multiplicity of an internal knot value is ('order' - 1), and the maximum multiplicity of an end knot value is 'order'.
- For periodic curves there must be ('nctrl' + 1) knot values, and the maximum multiplicity of any knot value is ('order' - 1). If the periodic knot has multiplicity greater than 1, repetitions must be given at the end of the knot vector.

Number of properties 'nprops':

- Gives the number of properties in the 'props' array.
- There is only one property (periodicity) and so 'nprops' must be either 0 or 1.

Properties array 'props':

- PAPRPE - the curve is periodic (default is non-periodic).

CRBSPS - Create B-surface from B-spline data

Receives

KI_int_dimension	*dim	dimension of control points
KI_int_order	*uorder	order of surface in u
KI_int_order	*vorder	order of surface in v
KI_int_nitems	*ncol	number of cols of control points
KI_int_nitems	*nrow	number of rows of control points
KI_dbl_coefficients	ctrl[dim * ncol * nrow]	control points
KI_dbl_knots	uknots[]	knot vector for the rows
KI_dbl_knots	vknots[]	knot vector for the columns
<KI_int_nitems>	*nprops	number of surface properties
KI_cod_papr	props[nprops]	array of surface properties

Returns

KI_tag_b_surface	*bs	B-surface
KI_cod_error	*ifail	failure indicator

Specific Errors

`KI_bad_knots`	`Invalid knot vector`
`KI_weight_le_0`	`weights are non positive`
`KI_insufficient_points`	`insufficient control points`
`KI_bad_parametric_prop`	`property not applicable`

Description This function creates a B-surface from B-spline data - i.e. a set of B-spline vertices (control points), and knot vectors for both the u and v directions. The surface may be either periodic or non-periodic, in either u or v, and the 'props' array supplies this information.

Dimension of control points 'dim':

- For rational surfaces 'dim'=4.
- For non-rational surfaces 'dim'=3.

Order of the surface in u and v respectively, 'uorder' and 'vorder':

- The order is the degree + 1.
- The minimum order is 2.

Number of columns of control points 'ncol':

- For surfaces with non-periodic rows 'ncol' >= 'uorder'.
- For surfaces with periodic rows 'ncol' >= 3.
- The number of control points in each row is 'ncol'.

Number of rows of control points 'nrow':

- For surfaces with non-periodic columns 'nrow' >= 'vorder'.
- For surfaces with periodic columns 'nrow' >= 3.
- The number of control points in each column is 'nrow'.

Control points 'ctrl':

- The control points are supplied row by row, each row containing 'ncol' vectors.
- For non-rational surfaces, the vectors are points in 3-space and must be supplied [x0,y0,z0,x1,y1,z1,...]
- For rational surfaces each vector contains a point in 3-space followed by a weight for the point. The points are supplied [x0,y0,z0,w0,x1,y1,z1,w1,...]. The weight must be positive. Increasing the weight of a control point will pull the surface closer to that point.

Knot vector for the rows 'uknots':

- The knot values must form a non-decreasing sequence.
- For non-periodic rows there must be ('ncol' + 'uorder') knot values, the maximum multiplicity of an internal knot value is ('uorder' - 1), and the maximum multiplicity of an end knot value is 'uorder'.
- For periodic rows there must be ('ncol' + 1) knot values, and the maximum multiplicity of any knot value is ('uorder' - 1). If the periodic knot has multiplicity greater than 1, repetitions must be given at the end of the knot vector.

Knot vector for the columns 'vknots':

- The knot values must form a non-decreasing sequence.
- For non-periodic columns there must be ('nrow' + 'vorder') knot values, the maximum multiplicity of an internal knot value is ('vorder' - 1), and the maximum multiplicity of an end knot value is 'vorder'.
- For periodic columns there must be ('nrow' + 1) knot values, and the maximum multiplicity of any knot value is ('vorder' - 1). If the periodic knot has multiplicity greater than 1, repetitions must be given at the end of the knot vector.

Number of properties 'nprops':

- Gives the number of properties in the 'props' array.
- There are only two properties (periodicity of rows and columns) and so 'nprops' must be either 0, 1 or 2.

Properties array 'props':

- PAPRPU - the surface is periodic in u (i.e. the rows are periodic).
- PAPRPV - the surface is periodic in v (i.e. the columns are periodic).

In both cases, the default is non-periodic.

CRBXSO - Create box solid

Receives

```
KI_vec_position          centre    centre of base of box
KI_vec_axis              axis      axis of box
KI_dbl_distance         *width     width of box
KI_dbl_distance         *length    length of box
<KI_dbl_distance>       *height    height of box
```

Returns

```
KI_tag_body             *box1      box
KI_cod_error            *ifail     failure indicator
```

Description If 'height' is positive, a body is created with six rectangular faces (a box or cuboid). One of the faces is centered at the point 'center', and the body extends a distance 'height' from there in the direction of 'axis'. The other two dimensions of the box are 'width' and 'height'.

If 'height' is zero, a rectangular sheet body is created, of dimension 'width' by 'length'. The normal to the plane face of the sheet is 'axis'.

In either case, if 'axis' is parallel to the Z-axis (0,0,1) then the faces or sides of length 'width' will be parallel to the X-axis, and those of length 'length' parallel to the Y-axis. If 'axis' is not parallel to the Z-axis, the orientation of the body about 'axis' is not defined.

CRBYGE - Creates a body from geometry

Receives

```
KI_tag_geometry  *geom                 curve or surface
<KI_int_nitems>  *nopts                number of options supplied
KI_cod_cbop       popts[nopts]         array of options
<KI_tag_list>     pdata[nopts]         array of option data
```

Returns

```
KI_tag_body      *body               wire or sheet body
KI_cod_error     *ifail              failure indicator
```

Specific Errors

```
KI_cant_extract_geom   failed to extract geometry
KI_bad_parameter       parametric limits not valid on this geometry
KI_non_manifold        resulting body would be non manifold
KI_bad_shared_dep      dependent of entity would be illegally shared
KI_unsuitable_entity   unsuitable entity
KI_is_attached         geometric entity is attached to topology
KI_invalid_geometry    geometry fails to pass checks
KI_bad_parametric_prop inappropriate property
KI_bad_tag_in_list     invalid null tag
```

Description This function creates a body corresponding to a geometric entity. The geometric entity may be any curve or surface type with the exception of a blending surface. If the geometric entity is a curve, a wire will be made; if it is a surface then a sheet will be made. The wire or sheet may be closed. The geometric entity must be capable of passing the checks imposed by CHCKEN.

The geometric entity must not be a dependent of another entity.

The options allow the user to supply parameter range information so that a body may be made up from part of a geometric entity. If no options are supplied then the entire parameter range/ranges is/are used. Note that for infinite geometric entities, such as planes and lines, this will fail. Options are not permitted if the geometric entity is a trimmed curve.

The property tokens are CBOPUR and CBOPVR, meaning the u and v parameter ranges respectively. For curves only CBOPUR may be used. If either or both are given then for each a list of two doubles must be supplied as data, specifying the appropriate parameter range. The lists should appear in the same order as the corresponding tokens.

For each pair of parameter limits, the following rules apply:

- The first element must be less than the second.
- Both elements must lie inside the parameter range, as given by ENCUPA/ENSUPA, unless the corresponding parameter is periodic. In that case the first must lie in the range, and the difference between the two may not exceed the period. The resulting wire or face will straddle the boundary of the parametrisation.

CRBYGE will seek to make the topology and geometry of the resulting body conform to the `special checks on geometry attached to topology' rules specified in CHCKEN by splitting faces and edges into smaller pieces so that they do pass these checks. These rules on attachability of geometry are affected in the same way by the value of SLIPCO as outlined in the CHCKEN documentation. Note that when a face or edge is split, its underlying geometry is also explicitly split.

The self intersection check is only performed if SLIPSI (see SEINTP) is set to a non zero value.

CRCAPO - Create a cartesian point

Receives

```
double              *x          X coordinate value
double              *y          y coordinate value
double              *z          Z coordinate value
```

Returns

```
KI_tag_point        *point      cartesian point
KI_cod_error        *ifail      failure code
```

Specific Errors

```
KI_bad_position     point must lie within the size box
```

Description A point is created with the given coordinates.

CRCICU - Creates a circular curve

Receives

```
KI_vec_centre         centre      centre of circle
KI_vec_axis           axis        axis direction
KI_dbl_radius        *radius      radius of circle
```

Returns

```
KI_tag_curve         *circle      circular curve
KI_cod_error         *ifail       failure code
```

Specific Errors

```
KI_radius_too_large            radius is too large
```

Description A complete circle is created with the given centre, radius, and axis direction. The sense of the circle is clockwise when looking along the axis direction.

CRCMPC - Join B-curves into a single curve

Receives

```
KI_int_nitems        *nbcs           number of B-curves
KI_tag_b_curve        bcs[nbcs]      array of B-curves
```

Returns

```
KI_tag_b_curve       *bc             B-curve
KI_cod_error         *ifail          failure indicator
```

Specific Errors

```
KI_curves_dont_meet         curves are not sequential
KI_bad_curves               invalid curves for joining
KI_insufficient_curves      insufficient curves to join
```

Description This function creates a B-curve by joining together a sequence of B-curves.

- At least two B-curves must be supplied ('nbcs' >= 2).
- The supplied curves must be in the correct order, and the end of each curve (except the last) must coincide with the start of the next. The curves must not be closed.
- The end of the final curve may coincide with the start of the first, in which case a closed B-curve will be made.
- The supplied curves are unchanged by the operation.
- The order of the resulting curve will be the maximum of the orders of the supplied curves.
- If any of the supplied curves are rational, the resulting curve will be rational.

CRCOSO - Create conical solid

Receives

```
KI_vec_position        centre    centre of base of cone
KI_vec_axis            axis      axis of cone
<KI_dbl_radius>       *basrad    radius of base of cone
<KI_dbl_radius>       *toprad    radius of top of cone
KI_dbl_distance       *height    height of cone
```

Returns

```
KI_tag_body           *cone      cone
KI_cod_error          *ifail     failure indicator
```

Specific Errors

```
KI_cone_too_sharp   Cone cannot be distinguished from a cylinder
KI_radii_both_0     Both radii are zero
KI_radius_too_large Top radius too large, base radius too large
```

Description A (truncated) cone of the given dimensions is created, its base centered at the given point and aligned along the given axis. The cone will be truncated unless one of the radii is zero.

CRCOSU - Create a conical surface

Receives

```
KI_vec_centre       posn     position on cone axis
KI_vec_axis         axis     axis direction
<KI_dbl_radius>    *radpsn   radius of cone at 'posn'
KI_dbl_angle       *angle    half-angle of cone ( radians )
```

Returns

```
KI_tag_surface     *cone     conical surface
KI_cod_error       *ifail    failure code
```

Specific Errors

```
KI_radius_lt_0                 radius negative
KI_bad_angle                   cone angle incorrect
KI_radius_too_large            invalid radius value
```

Description A conical surface is created according to the parameters specified.

The mathematical definition of a cone defines two half cones, apex to apex on a common axis. Only one of the pair is created by this routine.

The half cone is fixed in space by 'axis', the axis direction; 'posn', a position on the cone axis and 'radpsn', the radius of the cone at 'posn'. 'radpsn' may be given as zero, so that 'posn' is the cone apex.

The sign of 'angle' determines which of the two possible half cones through the circle defined by 'posn', 'axis' and 'radius' is created.

Sign of Angle	Axis Direction
positive	towards apex
negative	away from apex

Parasolid will, however, convert the cone representation internally to one with a positive angle, with axis pointing towards the apex.

The surface normal points away from the axis.

CRCPCU - Creates a constant parameter line curve on the surface

Receives

```
KI_tag_surface     *surf    Underlying surface
KI_cod_papr        *uorv    Constant parameter
KI_dbl             *param   Constant parameter value
```

Returns

```
KI_tag_curve       *curve   Constant parameter line curve
KI_cod_error       *ifail   Failure indicator
```

Specific Errors

```
KI_wrong_entity         Unsupported surface type
KI_not_on_surface       Parameter out of range
KI_at_singularity       Curve not possible at parametric singularity
KI_invalid_geometry     Curve not possible
KI_bad_parametric_prop  Inappropriate property
```

Description This function creates a curve corresponding to a constant parameter line on the surface.

The type of the curve returned is simplified wherever possible. However, when the supplied surface is a foreign geometry surface (type TYSUFG) then the type of the resulting extracted curve will be TYCUCP and when it is an offset surface (type TYSUOF) then the resulting extracted curve may be of type TYCUSP.

In some cases the returned constant parameter curve may extend beyond the confines of the surface. This case is most likely where the returned curve is simplified to a curve which contains the constant parameter curve as a subset. An example of this would be a line of constant u parameter on a lemon torus. This constant parameter curve would be returned as a complete circle.

Argument	Meaning
`surf'	is the existing surface from which the curve is extracted.

`uorv'	defines whether the curve is : Constant u (uorv = PAPRUP), Constant v (uorv = PAPRVP).
`param'	supplies that constant parameter value. If ENSUPA indicates that 'surf' is not periodic or infinite in the parameter 'uorv', the value of 'param' must lie within the range specified by ENSUPA for 'surf'.
`curve'	is the returned tag of the curve, if the operation succeeds.

It is not possible to extract constant v parameter curves (PAPRVP) from blend surfaces (type TYSUBL).

If 'surf' is not an orphan, then the constant parameter curve will be created as construction geometry in the owning part of 'surf'.

CRCUPC - Create B-curve from general curve

Receives

```
KI_tag_curve         *curve        general curve
KI_vec_position       bounds[2]    start and end of curve
```

Returns

```
KI_tag_b_curve       *bc           B-curve
KI_int_nitems        *nseg         number of segments used
KI_cod_error         *ifail        failure indicator
```

Specific Errors

```
KI_bad_end_points  curve not defined between bounds
KI_coincident      coincident bounds on non-periodic curve
KI_not_on_curve    end bound not on curve, start bound not on curve
KI_wrong_entity    unsupported curve type
```

Description This function creates a B-curve that is equivalent to a general curve, so that the curve can be used for example in lofting. Only lines, circles, ellipses and B-curves are processed by this function. Trimmed curves with the aforementioned curves as basis curves are also allowed. The B-curve returned will exactly represent the given curve.

The function may be used to trim curves where a B-curve is required otherwise CRTRCU should be used to trim curves.

The following table shows the type of curve that will be produced by the conversion:

Original Curve	Order of New Curve	Number of Segments	Rational
line	2	1	no
circle	4	1 or 2	yes
ellipse	4	1 or 2	yes
B-curve	as original	betweens bounds on original	as original

The bounds of the curve 'bounds' must be supplied in order, start then end. The new curve will have a parametrisation increasing from start to end. If the curve is a trimmed curve the bounds are ignored and any valid vectors should be supplied.

CRCYSO - Create cylindrical solid

Receives

```
KI_vec_position      centre          centre of base of cylinder
KI_vec_axis          axis            axis of cylinder
KI_dbl_radius       *radius          radius of cylinder
<KI_dbl_distance>   *height          height of cylinder
```

Returns

```
KI_tag_body         *cylind          cylinder
KI_cod_error        *ifail           failure indicator
```

Specific Errors

```
KI_radius_too_large                Radius too large
```

Description If 'height' is positive, a body is created which is a right circular cylinder of length 'height' and radius 'radius'. One end face is centered at 'centre', and the axis of the cylinder is parallel with 'axis'.

If 'height' is zero, a sheet body is created which is a circle, center 'centre', radius 'radius'. The normal to the plane of the sheet is parallel to 'axis'.

CRCYSU - Create a cylindrical surface

Receives

```
KI_vec_position          posn       position on axis
KI_vec_axis              axis       axis direction
KI_dbl_radius           *radius     cylinder radius
```

Returns

```
KI_tag_surface          *cylind     cylindrical surface
KI_cod_error            *ifail      failure code
```

Specific Errors

```
KI_radius_too_large                radius too large
```

Description An unbounded cylindrical surface of radius 'radius' is created with an axis 'axis' which passes through 'posn'.

The surface normal points away from the axis.

CREASS - Create assembly

Receives

```
KI_cod_tyas          *type      type of assembly to create
```

Returns

```
KI_tag_assembly      *assemb    new assembly
KI_cod_error         *ifail     failure indicator
```

Description CREASS creates a new assembly and attaches it to the world. The new assembly has no instances and is not instanced. The box of the assembly will be unset. The state of the assembly will be new (ENSTNW) and it will have no key.

'type' must be a token of the form TYASxx. At present the only acceptable type of assembly is collective (TYASCL).

CREATT - Create an attribute

Receives

```
KI_tag_list_entity *owners         entity or list of entities to which
                                   the attribute is to be attached
KI_tag_attrib_def  *type           type of attribute
<KI_int_nitems>    *nints          number of integer values
int                 ivals[nints]   array of integer values
<KI_int_nitems>    *nreals         number of real values
double              rvals[nreals]  array of real values
<KI_int_nitems>    *nstrng         number of strings
<KI_int_nitems>     slens[nstrng]  array of string lengths
<KI_int_nitems>    *nchars         number of characters
KI_chr_string       chars[nchars]  array of data for string fields
```

Returns

```
KI_cod_error      *ifail        error code
```

Specific Errors

```
KI_wrong_entity_in_list   entity in list cannot own given attribute
KI_system_error           no attribute
KI_attr_type_not_defined  not a valid attribute definition
KI_attr_mismatch          insufficient data for an attribute of
                          this type
KI_buffer_overflow        array 'chars' is too small
```

Description An attribute of the given type is created, loaded with the given data and attached to each entity in the list 'owners'.

The entities in 'owners' must be of types specified as legal owners for attributes of the given 'type'. The legal owners of a type can be determined by calling OUATDF.

The values to go into the fields are supplied in three arrays. Each field is filled in turn by taking the next value from the array of the appropriate type.

Thus the number of real values supplied, 'nreals', is the number of real fields plus three times the number of direction, vector and co-ordinate fields plus six times the number of axis fields.

If there are any character string fields in the attribute their values are supplied, concatenated, in the array ('chars'). The number of strings is passed in 'nstrng' and the array 'slens' contains the string lengths.

If 'nchars' is not greater than or equal to the sum of the values in the array 'slens', KI_buffer_overflow is returned in 'ifail' and no attributes are created.

If an attribute of the given type is already attached to an entity in 'owners', the old attribute is replaced by one containing the given data, for attributes of classes RQAC01 to RQAC05. For attributes of class RQAC06 and RQAC07, the list of attribute values is extended to contain the new value.

CREFEA - Create a feature

Receives

```
KI_cod_tyfe         *type      type of feature
KI_tag_part         *part      part which will own feature
```

Returns

```
KI_tag_feature      *featre    newly created feature
KI_cod_error        *ifail     failure code
```

Specific Errors

```
KI_wrong_type_for_feat  feature type cannot be created in 'part'
```

Description Creates an empty feature in the specified part.

Features created in assemblies may have types:

Token	Feature
TYFEIN	instance
TYFESU	surface
TYFECU	curve
TYFEPT	point
TYFEMX	mixed

If any but TYFEMX is given, the feature will be permitted to contain entities only of the specified type. A mixed feature in an assembly can contain entities of any of the four types.

Items can be contained in a feature in an assembly only if they belong to the assembly. Surfaces, curves and points must be construction geometry attached to the assembly. Instances must be owned by the assembly.

Features created in bodies may have types:

Token	Feature
TYFERG	region
TYFEFA	face
TYFEED	edge
TYFEVX	vertex
TYFESU	surface
TYFECU	curve
TYFEPT	point
TYFEMX	mixed

If any but TYFEMX is given, the feature will be permitted to contain entities only of the specified type. A mixed feature in a body can contain entities of any of the seven types.

Items can be contained in a feature in a body only if they are contained within the body. Surfaces, curves and points may be either construction geometry attached to the body or be attached to a face, edge or vertex in the body. Regions, faces, edges and vertices must be contained within the body.

CREINS - Create instance

Receives

```
KI_tag_assembly      *owner   owning assembly of new instance
KI_tag_part          *part    part to be instanced
<KI_tag_transform>   *transf  transform of instance
KI_cod_tyin          *type    type of instance
```

Returns

```
KI_tag_instance      *instnc  new instance
KI_cod_error         *ifail   failure indicator
```

Specific Errors

```
KI_wrong_transf          transform contains scale or reflection
KI_anon_sub_part         instance of anonymous sub-part of stored part
KI_cyclic_assy           instance would cause cyclic reference
KI_not_in_same_partition owner and part are in different partitions
```

Description CREINS creates a new instance of 'part' and attaches it to the assembly 'owner'.

If the new instance would cause the assembly graph to become cyclic (i.e. when creating an instance of P in A, and P or a sub-part of P is A) KI_cyclic_assy will be returned in 'ifail'.

The part may not be unloaded (ENSTUN).

We define the true-sub-parts of a stored (ENSTST) part S as those anonymous (ENSTAN) sub-parts of S reachable from S without encountering other stored parts (see OUPART for a description of all part states).

If P is anonymous and is a true-sub-part of some stored part S then P may only be instanced from S or a true-sub-part of S. If this condition is not met KI_anon_sub_part is returned in 'ifail'.

A copy of 'transf' is attached to the new instance.

If a null tag is given for the transform, the transform field of the instance is left null and the system will act as if the instance has an identity transform attached.

The transform must not be a general affine transformation and must consist of translation and rotation only: if scaling, reflection or shearing is present KI_wrong_transf is returned in 'ifail'.

The type given must a token in the range TYINxx. At present the only valid value is positive (TYINPS).

Creating an instance is regarded as changing the owning assembly of the new instance but not the part that is instanced. If the assembly is new or modified no change of states will take place. If it is anonymous or stored its state will be changed to new and modified respectively. The effect of this change will be rippled up the parts graph.

CRELCU - Create an elliptic curve

Receives

```
KI_vec_centre           centre      centre of ellipse
```

```
KI_vec_axis               axis       axis direction
KI_dbl_radius            *majrad     major radius
KI_vec_axis               majaxi     major axis
KI_dbl_radius            *minrad     minor radius
```

Returns

```
KI_tag_curve             *elipse     elliptical curve
KI_cod_error             *ifail      failure code
```

Specific Errors

```
KI_majrad_minrad_mismatch  major radius less than minor radius
KI_majaxi_not_perpn        major axis and axis not perpendicular
KI_radius_too_large        radius too large
```

Description An elliptical curve is created. The direction of the major axis must be at right angles to the axis of the ellipse. The minor radius must be less than or equal to the major radius.

The curve has a clockwise direction, when viewed along the axis direction.

CREQSC - Create an equal scaling transformation

Receives

```
KI_dbl_sc_fact       *scale    scaling factor
KI_vec_position       centre   centre of scaling
```

Returns

```
KI_tag_transform     *transf   equal scaling transformation
KI_cod_error         *ifail    failure code
```

Description Creates a transformation matrix that, when applied to an entity, causes an equal scaling along all axes, centered on the given position.

CREREF - Create a reflection transformation

Receives

```
KI_vec_position        posn     position on plane
KI_vec_normal          normal   normal direction
```

Returns

```
KI_tag_transform      *transf   reflection transformation
KI_cod_error          *ifail    failure code
```

Description Calculates a transformation matrix that, when applied to an entity, causes it to be reflected in the plane defined by the given position and normal.

CREROT - Create a rotation transformation

Receives

```
KI_vec_position        posn      position on axis
KI_vec_axis            axis      axis direction
KI_dbl_angle          *angle     rotation angle (radians)
```

Returns

```
KI_tag_transform      *transf    rotation transformation
KI_cod_error          *ifail     failure code
```

Specific Errors

```
KI_rot_angle_eq_0        zero angle rotation asked for
```

Description Calculates a transformation matrix that, when applied to an entity, rotates it about the axis which passes through the given position. The angle of rotation (in radians) must not be within the resolution angle of zero. A 'right-hand screw rule' is assumed when calculating rotational direction with respect to axis direction.

CRETFM - Creates a general transformation from the given matrix

Receives

```
double                matrix[16]    transformation components
```

Returns

```
KI_tag_transform     *transf        the transformation
KI_cod_error         *ifail         failure code
```

Specific Errors

```
KI_wrong_transf      determinant is zero
KI_sc_factor_le_0    scale must be greater than zero
KI_bad_component     array positions 12, 13, 14 must be zero
```

Description The array 'matrix' contains the components that make up the transformation.

The matrix operates as a post-multiplier on row vectors containing homogenous coordinates thus:

```
(x',y',z',s') = (x,y,z,s) T
```

where the conventional 3-d coordinates are

```
(x/s,y/s,z/s).
```

The matrix thus consists of

```
(              , 0 )
(      R       , 0 )
(              , 0 )
( Tx,  Ty,  Tz,   S )
```

R = a non singular transformation matrix. This matrix contains the rotation, reflection, local scaling and shearing components

T = a translation vector

S = a scaling factor. It has to be greater than zero.

'matrix' should be filled as follows:

Positions in Array	**Contents**
0 through 2	transformation element r_{11} r_{21} r_{31}
4 through 6	transformation element r_{12} r_{22} r_{32}
and 8 through 10	transformation element r_{13} r_{23} r_{33}

3, 7 and 11	translation vector
12 through 14	0.0
15	scale

Note: Only rotation and translation are permitted in transformations attached to instances.

Transformations where R is non-orthogonal (contains shearing or local scaling) cannot be applied to many entities (see APPTRA).

CRETRA - Create a translation transformation

Receives

```
KI_vec_direction       direct   direction
KI_dbl                *dist     distance
```

Returns

```
KI_tag_transform      *transf   translation transformation
KI_cod_error          *ifail    failure code
```

Description Calculates a transformation matrix that, when applied to an entity, causes it to be translated in the given direction vector by the given distance.

CREXSU - Create an extruded surface

Receives

```
KI_tag_curve          *profil   curve to be extruded
KI_vec_direction       direct   extrusion direction vector
KI_cod_logical        *smplfy   simplification flag
```

Returns

```
KI_tag_surface        *extsur   resulting extruded surface
KI_cod_error          *ifail    error code
```

Specific Errors

```
KI_impossible_sweep      Cannot determine extruded geometry
KI_unsuitable_entity     Not one of the allowed curve types
KI_invalid_geometry      Invalid curve
```

Description This function creates an extruded or swept surface.

The curve 'profil' is extruded along the direction vector 'direct', the new surface being the envelope of the curve.

If 'smplfy' is KI_false then the function will always return a surface of type TYSUSE. If 'smplfy' is KI_true then, if possible, the function will return an analytic surface which is equivalent to the extruded surface. For example, an extruded line is almost always equivalent to a plane.

The curve must be of one of the following types:

- line
- circle
- ellipse
- B-curve

A self intersecting surface may be returned, but it is not possible to sweep a line along its own direction vector.

If 'profil' is not an orphan, the resulting surface will be created as construction geometry in the owning part. A curve which is a dependent of another entity may be extruded - for example, a single curve may be the underlying curve of two swept surfaces.

CRFASU - Create surface to fit and attach to face

Receives

```
KI_tag_face        *face      face to be fixed
```

Returns

```
KI_cod_tysu        *sutype    type of surface fixed to face
KI_tag_surface     *surfac    new surface fixed to face
KI_cod_rtlo        *state     state of the body
                               RTLOOK => Valid
                               RTLONG => Negated
                               RTLOSX => Self-Intersecting
KI_cod_error       *ifail     failure code
```

Specific Errors

```
KI_cant_find_su      Unable to find a surface
KI_unsuitable_entity Face does not belong to a sheet or solid body
KI_general_body      General body
```

Description If possible, a surface is created, consistent with the edge geometry of the face, which replaces the existing surface geometry (typically rubber).

If a simple surface (plane, cylinder, torus, cone or sphere) will fit, it will be used; if no simple surface will fit, the modeler may be able to fit another surface (such as a blend or B-surface) but this is not guaranteed. It may be that more than one surface will fit the edges. In this case it is not defined which surface will be used.

In general this procedure may give rise to self-intersecting object boundaries which could cause unpredictable errors later.

If local checking is on, (see SEINTP and OUINTP) consistency checks will be made on newly created topological and geometrical entities, and the state of the body returned. A state of RTLOOK indicates the body is valid. A state of RTLONG indicates that the result body was originally "inside out" but has been negated, and is now "positive" (has positive volume) and valid. A state of RTLOSX indicates the body is self-intersecting and further modelling operations on it may fail. It is the responsibility of the calling routine to make any necessary reparation.

If the session parameter for local checking is switched off, the state returned will be RTLOOK regardless.

A self-intersecting body can be returned even if the ifail is zero.

This function should only be called for faces in sheet or solid bodies. It is not supported for general bodies.

CRFGCU - Creates a Foreign Geometry curve

Receives

```
KI_int_nchars    *keylen         Length of curve key
KI_chr_key        key[keylen]    Curve key
KI_int_nitems    *nspace         Space required by foreign curve
                                 (specified in doubles)
<KI_int_nitems>  *nints          Number of integer values
int               ivals[nints]   Array of integer values
<KI_int_nitems>  *nreals         Number of real values
double            rvals[nreals]  Array of real values
```

Returns

```
KI_tag_curve     *curve          Curve
KI_cod_error     *ifail          Failure indicator
```

Specific Errors

```
KI_FG_evaluator_error      Curve evaluator failure
KI_FG_modelling_error      Cannot model with this curve
KI_FG_data_not_found       Curve evaluator data not found
KI_FG_integer_data_error   Curve evaluator integer data error
KI_FG_real_data_error      Curve evaluator real data error
KI_FG_data_alloc_error     Curve evaluator data allocation fault
KI_FG_evaluator_not_found  Curve evaluator not found
```

Description This function creates a foreign curve, i.e. one whose evaluator is external to Parasolid. This function call will only produce a curve if an appropriate curve evaluator has been linked to Parasolid. The curve is identified by its key. Any additional data required by the evaluator is supplied in arrays containing integer and real values.

This KI call will result in the execution of the initialization routines of the FG evaluator system. Function arguments are as follow:

Argument	Meaning
`keylen'	is the length or number of characters contained within the 'key'.
`key'	to uniquely identify the curve evaluator.
`nspace'	specifies the length of the block of data to be initialized by the foreign curve loader.
`nints', `ivals', `nreals', `rvals'	specify the numerical data supplied to the curve evaluator.
`curve'	is the returned tag of a foreign curve, if the operation succeeds.

CRFGSU - Creates a foreign surface

Receives

```
KI_int_nchars    *keylen         Length of surface key
KI_chr_key        key[keylen]    Surface key
KI_int_nitems    *nspace         Space required by foreign surface
                                 (specified in doubles)
```

```
<KI_int_nitems> *nints          Number of integer values
int              ivals[nints]   Array of integer values
<KI_int_nitems> *nreals         Number of real values
double           rvals[nreals]  Array of real values
```

Returns

```
KI_tag_surface  *surf           surface
KI_cod_error    *ifail          failure indicator
```

Specific Errors

```
KI_FG_evaluator_error        Surface evaluator failure
KI_FG_modelling_error        Cannot model with this surface
KI_FG_data_not_found         Surface evaluator data not found
KI_FG_integer_data_error     FG integer data error
KI_FG_real_data_error        FG real data error
KI_FG_data_alloc_error       FG data allocation fault
KI_FG_evaluator_not_found    Surface evaluator not found
```

Description This function creates a foreign surface, i.e. one whose evaluator is external to Parasolid. This function call will only produce a surface if an appropriate surface evaluator has been linked to Parasolid. The surface is identified by its key. Any additional data required by the evaluator is supplied in arrays containing integer and real values.

This KI call will result in the execution of the initialization routines of the FG evaluator system. Function arguments are as follow:

Argument	Meaning
`keylen'	is the length or number of characters contained within the 'key'.
`key'	to uniquely identify the curve evaluator.
`nspace'	specifies the length of the block of data to be initialized by the foreign curve loader.
`nints', `ivals', `nreals', `rvals'	specify the numerical data supplied to the curve evaluator.
`surf'	is the returned tag of a foreign surface, if the operation succeeds.

CRINCU - Create intersection curves

Receives

```
KI_tag_surface             *surf1      surfaces to be
KI_tag_surface             *surf2      intersected
KI_dbl_box                  intbox[6]  box of interest
```

Returns

```
<KI_tag_list_curve>        *curves     list of curves
<KI_int_nitems>            *ncurve     number of curves returned
KI_cod_error               *ifail      failure code
```

Specific Errors

```
KI_cant_do_intersect      failure in intersection routine
KI_su_are_coincident      coincident surfaces
KI_dont_intersect         no intersection
KI_invalid_geometry       surface fails checks
```

Description New curves are created wherever the two surfaces intersect (if they do). The list of new curves is returned in 'curves'. All curves which pass through the box of interest will be returned; curves which lie completely outside the box may be returned. Curves may be truncated more or less immediately when they leave the box, and a curve which leaves and reenters the box may be returned as several disjoint sections.

'intbox' describes the box that contains the area of interest and is specified as follows.

Element	Contents
1st	minimum x_component of the area of interest
2nd	minimum y_component of the area of interest
3rd	minimum z_component of the area of interest
4th	maximum x_component of the area of interest
5th	maximum y_component of the area of interest
6th	maximum z_component of the area of interest

For a box to be valid the difference between the maximum and minimum components in all three principal directions must be greater than or equal to zero.

If either surface has type TYSUPA (B-surface) then that surface must be capable of passing the checks imposed by CHCKEN.

The self intersection check is only performed if the appropriate option is set (see SEINTP).

CRKNPA - Creates a knitting pattern from a list of bodies

Receives

```
KI_tag_list_body    *bods   list of bodies
```

Returns

```
<KI_tag_list_edge> *eds1   list of edges forming pattern
<KI_tag_list_edge> *eds2   list of edges forming pattern
<KI_int_nitems>    *neds   number of edge pairs in pattern
<KI_tag_list_body> *negs   list of bodies to be negated
<KI_int_nitems>    *nnegs  number of bodies in 'negs'
<KI_tag_list_body> *over   list of bodies with no edges in the pattern
<KI_int_nitems>    *nover  number of bodies in 'over'
KI_cod_error       *ifail  failure code
```

Specific Errors

```
KI_cant_create_pattern   failure to create knitting pattern
KI_pattern_invalid       knitting would result in invalid body
KI_bodies_dont_knit      no coincident edges exist
KI_bad_type              body in list is of incorrect type
KI_duplicate_list_item   duplicate item in list
KI_general_body          general body
```

Description This function creates two lists of edges which form the knitting pattern used as input to KNITEN. Corresponding elements in the lists are coincident edges that will be fused in the resultant knitted body.

The received body list 'bods' may contain either solid or sheet bodies, not wires. In the case of solids, all edges which the application intends to appear in the knitting pattern must have one adjacent face with a surface attached and one without. The faces in the solid without surfaces attached will be those which are eventually redundant when the resultant solid body is created using the pattern output by CRKNPA by calling KNITEN.

'eds1' and 'eds2' are the edge lists that form the knitting pattern and 'neds' is the length of each of these lists.

'negs' is a list of the bodies in the received list 'bods' that should be negated before calling KNITEN. This is done by calling NEGENT. If these bodies are not negated the knitting operation will result in inconsistent face normals within shells of the knitted body.

'nnegs' is the number of bodies in 'negs'.

'over' contains a list of leftover bodies, i.e. bodies which have no edges in the returned knitting pattern.

'nover' is the number of bodies in 'over'.

Negating some of the received bodies may not be sufficient to produce a valid knitted body. For example, knitting could result in a moebius strip. Such cases will be identified and will result in ifail KI_pattern_invalid.

If none of the edges in the received bodies are coincident, ifail KI_bodies_dont_knit will be returned. If however a subset of bodies in the received list has coincident edges, a knitting pattern will be created and an ifail will not be invoked. The bodies with no edges in the pattern will be returned in 'over'.

> **Note:** This routine may not preserve the topology of the received bodies: edges may be split in order to produce a 1:1 correspondence between the edges that form the pattern.

This function is not supported for general bodies.

CRLFPS - Create B-surface by lofting

Receives

```
KI_int_nitems      *nbcs            number of curves supplied
KI_tag_b_curve      bcs[nbcs]       array of B-curves
<KI_int_nitems>    *nprops          number of surface properties
KI_cod_papr         props[nprops]   array of surface properties
<KI_tag_list>       pdata[nprops]   array of tags of data lists
```

Returns

```
KI_tag_b_surface   *bs              B-surface
KI_cod_error       *ifail           failure indicator
```

Specific Errors

```
KI_wrong_number_knots  wrong number of knots in lofting knot vector
KI_bad_knots           curve knot vectors are incompatible, bad lofting
                       knot vector
KI_incompatible_curves curves cannot be matched
KI_bad_curves          cannot loft coincident curves
KI_insufficient_curves insufficient curves to loft
```

KI_bad_index	index for degenerate segment out of range
KI_bad_derivative	derivative too large, twist vector too large, wrong number of coordinates in twist vector
KI_wrong_number_derivs	wrong number of derivatives
KI_incompatible_props	incorrect use of degenerate segment property, incorrect use of amalgamate property, incompatible degenerate conditions, incompatible clamping conditions, incompatible boundary conditions
KI_bad_parametric_prop	inappropriate property
KI_bad_tag_in_list	invalid null tag in pdata list
KI_bad_degen_vertices	insufficient vertices for degenerate end curve, insufficient vertices for degenerate start curve
KI_bad_deriv_vertices	wrong number of deriv vertices for clamped end, wrong number of deriv vertices for clamped start

Description This function creates a B-surface by lofting through a set of B-curves.

Each curve shall lie along a constant v parameter line. The first curve shall lie along the v = 0 parameter line and the last curve shall lie along the v = 'nbcs'-1 parameter line. The parametrisation in the u direction is derived from the curves being lofted.

Continuity of lofted surface:

The surface is always continuous up to second order in the loft direction. If the lofted curves were splined or created as B-splines and they all share the same knot vector, then a surface is produced which is continuous in the curve direction to the same order as the definition curves. Otherwise the surface is, in general, only position continuous in the curve direction.

Dimension of surface:

The surface will be rational if one or more of the B-curves is rational, otherwise it is non-rational.

B-curves to loft 'nbcs', bpcs':

- If the surface is to be periodic in the loft direction (see below), at least three curves ('nbcs' >= 3) must be given.
- If the surface is to be non-periodic in the loft direction (see below), at least two curves ('nbcs' >= 2), or one curve and a degenerate start or end (see below) must be given.
- The curves must be in order in the list.
- The curves are unchanged by the operation.
- The curves must have the same number of segments, unless either the amalgamate option (see below) is used or the number of segments are equated by adding degenerate segments (see below).
- Consecutive curves must not be coincident.

Surface properties 'nprops', 'props', 'pdata':

There are several controls that may be applied to the lofting operation. For example, the surface may be periodic in the loft direction; a knot vector may be supplied, or the surface may degenerate to a point. Each action has a default, and each default can be overridden by giving a token in the 'props' array. 'nprops' is the number of tokens that have been supplied in 'props'.

A particular action may require additional data; if so, this must be supplied in a list. The tag of the list must be entered in the array 'pdata', in the position corresponding to the token in 'props'. If the action does not require additional data, then the null tag should be entered in the appropriate position in 'pdata'.

Property tokens:

The 'props' array contains 'nprops' tokens from the sequence PAPR00. The table shows which tokens may be used, and the data associated with them.

There are some pairs (or sets) of tokens which are alternatives; if both are supplied they may be contradictory, and in this case the last one to be supplied is the one which is used.

There are also cases in which the presence of a token implies a particular structure, and another implies a different structure. Use of both tokens is inconsistent, and raises an error.

In explaining the various controls that may be applied to the lofting operation the following notation is used:

- bottom left - on the surface at the start of the first curve
- bottom right - on the surface at the end of the first curve
- top left - on the surface at the start of the last curve
- top right - on the surface at the end of the last curve

Token	**Meaning**	**Data**
PAPRPE	surface is periodic in the loft direction	none
PAPRNS	no curvature across start curve i.e. natu ral boundary condition	none
PAPRNE	no curvature across end curve i.e. natu ral boundary condition	none
PAPRCS	derivatives supplied across start curve	vector at each spline point across i.e. clamped boundary condition
PAPRCE	derivatives supplied across end curve	vector at each spline point i.e. clamped boundary condition
PAPRSD	surface clamped with derivative curve at start	b-spline vertices of derivative curve
PAPRED	surface clamped with derivative curve at end	b-spline vertices of derivative curve
PAPRBL	bottom left twist vector supplied	twist vector
PAPRBR	bottom right twist vector supplied	twist vector
PAPRTL	top left twist vector supplied	twist vector
PAPRTR	top right twist vector supplied	twist vector
PAPRDS	surface is degenerate before start curve	degenerate point
PAPRDE	surface is degenerate after end curve	degenerate point
PAPRSW	degenerate curve supplied before start curve	b-spline vertices of degenerate curve

PAPREW	degenerate curve supplied after end curve	b-spline vertices of degenerate curve
PAPRKT	knot vector supplied	knot vector
PAPRAM	amalgamate option	none
PAPRIS	insert degenerate segment in start curve	segment position
PAPRIE	insert degenerate segment in end curve	segment position
PAPRCU	force cubic lofting	none

Boundary conditions PAPRPE, PAPRNS, PAPRNE, PAPRCS, PAPRCE:There are three boundary conditions available to control the lofting: natural, clamped and periodic. Natural and clamped conditions refer to either the start or end of the lofted surface, whereas the periodic boundary condition refers to both.

- Natural boundary conditions imply that the surface has no curvature in the loft direction, across the start or end curve as appropriate. Natural boundary conditions are the default.
- Clamped boundary conditions allow the user to specify derivatives across the start and end curves. There are two methods of supplying these derivatives. Options PAPRCS and PAPRCE may be used to clamp the surface with a list of derivative vectors. Similarly, options PAPRSD and PAPRED may be used to clamp the surface using a derivative b-spline curve.
  - If derivative vectors are used then they must be supplied at every spline point of the corresponding curve (a spline point is a point between adjacent segments of the curve, or the start or end of the curve). The derivative vectors must be stored in a real list of length (3 * 'nsp'), where nsp is the number of spline points on each curve.

    The derivatives should be supplied with respect to a parameter which varies from 0 to 1 between the first or last two curves (as appropriate). In other words, the derivatives should have dimensions of length. The magnitude is significant, and supplying vectors which are too large may cause the surface to loop or kink.
  - When a derivative b-spline is used to clamp an end of the surface it is only necessary to supply the control vertices. This is because the knot vector of the derivative curve will be the same as the knot vector of the corresponding start or end curve. The vertices should be supplied in a real list of length:

    ('dim' * 'ncrtl')

    where,

    dim = dimension of vertices on corresponding start or end curve

    nctrl = number of vertices on corresponding start or end curve

    A derivative b-spline provides a more complete definition of the start/end derivatives particularly when the curves to be lofted are either rational or contain multiple knots.

  If clamped boundary conditions are used with the insert degenerate segment option PAPRIS or PAPRIE then it should be noted that for every extra segment added the end curve will have one extra spline point and one extra control vertex. This means that if the end is clamped with derivative vectors one extra vector must be supplied for every segment added. Similarly if the surface is clamped with a derivative curve one extra control vertex is required for every segment added.

- Periodic boundary conditions imply that the surface is closed in the loft direction, so that the surface will return to the first curve in the array after the final one has been lofted. The surface will meet itself with continuity of tangent and curvature. If periodic boundary conditions are used, then at least three curves must be supplied ('nbcs' >= 3). Periodic boundary conditions cannot be used if either the first or last curve is degenerate.

Twist vectors PAPRBL, PAPRBR, PAPRTL, PAPRTR:

A twist vector is a derivative with respect to both u and v; i.e. it is the rate of change of the cross boundary derivatives in the direction of the curves. The twist vectors may be supplied at any of the four corners, but only when clamped boundary conditions have been supplied across the corresponding boundary. Each one should be supplied in a real list of length 3. If the twist vectors are supplied when they are not required then they are ignored. If the twist vectors are not supplied then a suitable default value is used.

Degenerate surface PAPRDS, PAPRDE, PAPRSW, PAPREW:

The lofted surface may degenerate to a point at either the start or the end of the loft. This degeneracy may be supplied either as a single point using PAPRDS and PAPRED or as a degenerate b-spline curve using PAPRSW and PAPREW. A degenerate b-spline curve will provide greater control when lofting curves that are rational.

- If the degeneracy is supplied as a point then the degenerate point must be given in a real list of length 3.
- When a degenerate b-spline curve is used to define the degeneracy it is only necessary to supply the control vertices. This is because the knot vector of the degenerate curve will be the same as the knot vector of the corresponding start or end curve. The vertices should be supplied in a real list of length :-

('dim' * 'ncrtl')

where,

dim = dimension of vertices on corresponding start or end curve

nctrl = number of vertices on corresponding start or end curve

Periodic end conditions cannot be used if either end of the lofted surface is degenerate, but it is possible to use clamped or natural end conditions. The insert degenerate segment options PAPRIS and PAPRIE cannot be used if the corresponding end of the surface is degenerate.

Knot vector PAPRKT:

If a knot vector is supplied then it must be given in a real list, and must satisfy the following conditions:

- The knot values must form an increasing sequence; repeated knots are not permitted.
- For non-periodic lofting there must be 'nbcs' knot values.
- For periodic lofting there must be ('nbcs'+1) knot values.

If the knot vector is not supplied then an averaged accumulated chord length parametrisation is used.

Amalgamation of knot vectors PAPRAM:

> An amalgamate property may be supplied for the surface, which will amalgamate the various knot vectors for the curves, and produce a surface which is continuous to the same order as the definition curves. However, this may produce a surface with a large number of patches, and should only be used on a small number of curves. The amalgamate option cannot be used in conjunction with inserted degenerate segments.

Insertion of degenerate segments PAPRIS, PAPRIE:

> If the first curve has fewer segments than the other curves PAPRIS may be used add the required number of segments to the first curve. Similarly PAPRIE may be used to add segments to the last curve. The segments added using these properties are degenerate, that is they contain the order of the curve coincident control points. Each degenerate segment added will cause the lofting process to make a patch with a corresponding degenerate boundary. Any number of degenerate segments may be added, by supplying the appropriate token more than once. An integer list containing one value must be supplied with each token; this specifies the segment preceding the required degenerate segment. If more than one degenerate segments are added they should be supplied in ascending order of segment number. Previously inserted degenerate segments affect the segment number of subsequent ones.

Degree of lofted surface PAPRCU:

> In general, the surface will be cubic in the loft direction. However, if only two non-rational curves are given and the boundary conditions are natural, then a ruled surface which is linear in the loft direction will be produced. The user may force the surface to be cubic in the loft direction by giving the PAPRCU property.

CRLICU - Create a linear curve (i.e. straight line)

Receives

```
KI_vec_position        posn      position on line
KI_vec_direction       direct    line direction
```

Returns

```
KI_tag_curve          *line      linear curve
KI_cod_error          *ifail     failure code
```

Description An unbounded straight line is created which passes through the given position in the given direction.

CRLIST - Create list entity

Receives

```
KI_cod_tyli  *lstype type of list to be created, from range TYLI00
```

Returns

```
KI_tag_list  *list   new list
KI_cod_error *ifail  failure indicator
```

Description An empty list entity of the given type is created.

The type must be one of:

Token	Type
TYLIIN	integer
TYLIRL	real
TYLITG	tag

Can be called from the GO.

CRMINO - Create minimum object

Returns

```
KI_tag_body          *minob      new minimum object
KI_cod_error         *ifail      failure indicator
```

Description CRMINO creates a new body and attaches it to the world. The body is a minimum object, with just one shell, consisting of one vertex. The only geometry created will be a single point at (0, 0, 0), attached to the vertex.

CROFSU - Create an offset surface

Receives

```
KI_tag_surface         *undrly     underlying surface
double                 *dstnce     offset distance
```

Returns

```
KI_tag_surface         *offsur     offset surface
KI_cod_error           *ifail      failure code
```

Specific Errors

```
KI_su_self_intersect    surface would be self-intersecting
KI_cant_offset          underlying surface cannot be offset
```

Description An offset surface is returned - this is defined to be the smooth, continuous (G1) surface obtained from the underlying surface by adding to each point on it the unit normal vector scaled by the offset distance. If the offset distance is zero then the underlying surface is returned, otherwise a new surface is created.

Whenever possible the surface created is not explicitly of type "Offset surface", but a simpler analytic type. In these cases CROFSU will not allow the caller to specify an offset distance such that the resulting surface would be inside out. For example, a sphere cannot be offset inwards by more than its radius.

For surfaces of type TYSUFG, the surface will only be created if the underlying surface is not degenerate.

For surfaces of type TYSUPA, TYSUSE, TYSUSU, the surface created will be checked for self intersections caused by degeneracies on the underlying surface. If such a self intersection is detected then an attempt will be made to trim the parameter range of the created surface to ensure that it is not self intersecting. If this fails then the surface will not be created.

> **Note:** If there are any degeneracies on the underlying surface, then the bound classification returned by ENSUPA for the underlying surface and for the created surface may be different.

No further checks for self intersection are made by CROFSU. However, the created offset surface may be checked for self intersection by calling CHCKEN with the appropriate option set (see CHCKEN and SEINTP).

If the SLIPCO interface parameter is zero, surfaces of type TYSUPA (B-surfaces) will be checked to ensure that they satisfy the composite geometry conditions which would be performed by CHCKEN if they were attached to a face. If this check fails then the offset surface will not be created.

The ifail KI_cant_offset is returned if the underlying surface is unsuitable for offsetting by any distance. The ifail KI_su_self_intersect is returned if the underlying surface is suitable but the requested offset is too large.

If 'undrly' is not an orphan, then the offset surface will be created as construction geometry in the owning part of 'undrly'.

Surfaces are not allowed to have more than one offset (of type TYSUOF). Consequently, 'offsur' may be created as an offset of a copy of 'undrly'.

> **Note:** If CROFSU creates a surface of type TYSUOF, the underlying surface of 'offsur' will be a dependent of it, with all the limitations that this implies. In particular, if the offset surface is deleted, then the underlying surface may also be deleted. (See DELENT)

CRPLSU - Create a planar surface

Receives

```
KI_vec_position          posn       position on plane
KI_vec_normal            normal     normal direction
```

Returns

```
KI_tag_surface          *plane      planar surface
KI_cod_error            *ifail      failure code
```

Description An unbounded plane is created passing through 'posn'. The surface normal is in the same direction as the given normal.

CRPRSO - Create prismatic solid

Receives

```
KI_vec_position        centre   centre of prism
KI_vec_axis            axis     axis of prism
KI_dbl_radius         *radius   radius of enclosing cylinder
int                   *nsides   number of sides
<KI_dbl_distance>     *height   height of prism
```

Returns

```
KI_tag_body           *prism    prism
KI_cod_error          *ifail    failure indicator
```

Specific Errors

KI_lt_3_sides	Number of sides is less than 3

Description If 'height' is positive, a body is created that is a right regular prism of 'nsides' sides. One of the end faces (regular polygons of 'nsides' sides) is centered at 'centre', and the prism extends a distance 'height' in the direction of 'axis'. The prism is inscribed in a cylinder of radius 'radius'.

If 'height' is zero, a sheet body is created whose face is a regular polygon of 'nsides' sides. The polygon is centered at 'centre', and the normal to the its plane face is parallel to 'axis'. Its circumscribing circle has radius 'radius'.

In either case, if 'axis' is parallel to the Z-axis (0,0,1) then one side (face or edge) will be perpendicular to the Y-axis, on the positive-Y side of 'centre'. If 'axis' is not parallel to the Z-axis, the orientation of the body about 'axis' is undefined.

CRPWPC - Create B-curve from piecewise data

Receives

KI_int_dimension	*dim	dimension of defining vectors
KI_int_order	*order	order of curve
KI_int_nitems	*nsegs	number of segments in curve
KI_dbl_coefficients	coeffs[dim * order * nsegs]	vectors defining the curve
KI_cod_slba	*basis	representation method

Returns

KI_tag_b_curve	*bc	B-curve
KI_cod_error	*ifail	failure indicator

Specific Errors

KI_discontinuous_curve	adjacent segments must meet
KI_weight_le_0	weights must be greater than zero
KI_bad_order	'order' must be four for Hermite basis

Description This function creates a B-curve from piecewise data. The following methods of representing the data are available:

- Bezier ('basis' = SLBABZ)
- Polynomial ('basis' = SLBAPY)
- Hermite (cubic only) ('basis' = SLBAHE)
- Taylor series ('basis' = SLBATA)

Dimension of coefficient vectors 'dim':

- For rational curves 'dim'=4.
- For non-rational curves 'dim'=3.

Order of each segment of the curve 'order':

- The order of the curve = degree + 1.
- The minimum order is 2.
- If the Hermite basis is used ('basis' = SLBAHE) then the curve has to be cubic ('order' = 4).

Number of segments in the curve 'nseg':

- There must be at least one segment ('nseg' >= 1).
- Adjacent segments must meet.

Coefficient data 'coeffs':

- Contains 'order'*'nseg' vectors of dimension 'dim'. If 'dim'=3, then the vectors are 3-D vectors giving the x, y and z components. If 'dim'=4, then each vector has a weight (w) associated with it, and x, y, z and w components are supplied for each vector. The weights supplied must be greater than zero.
- The coefficients are supplied in order, segment by segment.
- The interpretation of the coefficients depends on the representation method chosen; this is determined by the value of the argument 'basis'.

Representation method 'basis':

The expressions for each segment of the B-curve P(t) in the various representations are given below. For generality, the rational form is given. The simplification to the non-rational form can be obtained by setting both the weights and the denominator equal to 1.0.

- Bezier vertices SLBABZ:

  The equation of a rational Bezier curve segment is:

$$P(t) = \frac{\sum_{i=0}^{n} b_i(t) w_i V_i}{\sum_{i=0}^{n} b_i(t) w_i}$$

Where:

n = ‘order‘-1

V_1 = Bezier vertex

w_i = weight for V_i

$b_i(t)$ = Bezier coefficients, define by:

$$b_i(t) = \frac{n!}{i!(n-i)!} * t^i * (1-t)^{n-i}$$

The Bezier vertices are supplied V_0 ,w_0 ,...,V_n ,w_n for the rational form, or V_0,...,V_n for the non-rational form.

- Polynomial coefficients SLBAPY:

The curve equation is given by a rational polynomial of order `order':

$$P(t) = \frac{\sum_{i=0}^{n} w_i A_i t^i}{\sum_{i=0}^{n} w_i t^i}$$

Where:

$n = \text{'order'-1}$

$A_i = \text{Polynomial coefficient}$

$w_i = \text{weight for } A_i$

The polynomial coefficients are supplied starting with the constant term and ending with the term of highest degree.

- Hermite coefficients SLBAHE:

This method can only be used for cubics. The equation of the curve is:

$$P(t) = \frac{f0(t)\ w0\ P0 + g0(t)\ w1\ P1 + f1(t)\ d0\ D0 + g1(t)\ d1\ D1}{f0(t)\ w0 + g0(t)\ w1 + f1(t)\ d0 + g1(t)\ d1}$$

Where:

$f0(t) = 1 - 3t^2 + 2t^3$ $\qquad g0(t) = 3t^2 - 2t^3$

$f1(t) = t - 2t^2 + t^3$ $\qquad g1(t) = -t^2 + t^3$

$P0, P1 = \text{start and end points of segment}$

$D0, D1 = \text{derivatives at start and end}$

$w0, w1 = \text{weights at end points}$

$d0, d1 = \text{derivatives of weights at start and end}$

The coefficients are supplied as P0, w0, P1, w1, D0, d0, D1, d1 for the rational form, or P0, P1, D0, D1 for the non-rational form.

- Taylor series SLBATA:

This method stores the derivatives evaluated at the point start of each segment, allowing the curve to be reconstructed as a Taylor series:

$$P(t) = \frac{\sum_{i=0}^{n} \frac{w^{(i)} P^{(i)} t^{i}}{i!}}{\sum_{i=0}^{n} \frac{w^{(i)} t^{i}}{i!}}$$

Where:

n = ‘order‘-1

$P^{(i)}$ = i‘th derivative at t=0

$w^{(i)}$ = i‘th derivative of weight at t=0

The point is supplied first, followed by the 1st derivative and ending with the derivative of order 'order'-1.

CRPWPS - Create B-surface from piecewise data

Receives

KI_int_dimension	*dim	dimension of defining vectors
KI_int_order	*uorder	order of surface in u
KI_int_order	*vorder	order of surface in v
KI_int_nitems	*ncol	number of columns of patches
KI_int_nitems	*nrow	number of rows of patches
KI_dbl_coefficients	coeffs[dim * uorder * vorder * ncol * nrow]	vectors defining the surface
KI_cod_slba	*basis	representation method

Returns

KI_tag_b_surface	*bs	B-surface
KI_cod_error	*ifail	failure indicator

Specific Errors

KI_discontinuous_surface	adjacent patches must meet
KI_weight_le_0	weights must be greater than zero
KI_bad_order	'order' must be four for Hermite basis

Description This function creates a B-surface from piecewise data. The following methods of representing the data are available:

- Bezier ('basis' = SLBABZ)
- Polynomial ('basis' = SLBAPY)
- Hermite (cubic only) ('basis' = SLBAHE)
- Taylor series ('basis' = SLBATA)

Dimension of coefficient vectors 'dim':

- For rational surfaces 'dim'=4.
- For non-rational surfaces 'dim'=3.

Order of each patch of the surface in u 'uorder', and in v, 'vorder':

- The order of the surface = degree + 1.
- The minimum order is 2.
- If the Hermite basis is used ('basis' = SLBAHE) then the surface has to be bicubic ('uorder' = 'vorder' = 4).

Number of columns 'ncol':

- There must be at least one column.

Number of rows 'nrow':

- There must be at least one row.

Coefficient data 'coeffs':

- Contains ('uorder' * 'vorder' * 'ncol' * 'nrow') vectors of dimension 'dim'. If 'dim'=3, then the vectors are 3-D vectors giving the x, y and z components. If 'dim'=4, then each vector has a weight (w) associated with it, and x, y, z and w components are supplied for each vector. The weights supplied must be greater than zero.
- The data is supplied patch by patch, row by row.
- The interpretation of the patch data depends on the representation method chosen; this is determined by the value of the argument 'basis'.
- Adjacent patches (in the u and v directions) must meet all along the corresponding boundary; i.e. the surface must be continuous.

Representation method 'basis'

The expressions for each patch of the B-surface P(u,v) in the various representations are given below. For generality, the rational form is given. The simplification to the non-rational form can be obtained by setting both the weights and the denominator equal to 1.0.

- Bezier vertices SLBABZ:

  The equation of a rational Bezier surface patch is:

$$P(u, v) = \frac{\sum_{i=0}\sum_{j=0} b_i(u)b_j(v)w_{ij}V_{ij}}{\sum_{i=0}^{nu}\sum_{j=0}^{nv} b_i(u)b_j(v)w_{ij}}$$

Where:

$nu = \text{'uorder'}-1$

$nv = \text{'vorder'}-1$

$V_{ij} = \text{Bezier vertex}$

$w_{ij} = \text{weight for } V_{ij}$

$b_i(u), b(v)_j = \text{Bezier coefficients}$

For the rational form the Bezier vertices and weights are supplied:

$V_{00}, w_{00}, V_{10}, w_{10}, \ldots, V_{mo}, w_{m0}, V_{01}, w_{01}, \ldots, V_{m1}, w_{m1}, \ldots, V_{0n}, w_{0n}, \ldots, V_{mn}, w_{m}$

For the non-rational form the w's are missed out.

- Polynomial coefficients SLBAPY:

  The surface equation is given by a rational bi-polynomial of orders 'uorder', 'vorder':

$$P(u, v) = \frac{\sum_{i=0}^{nu}\sum_{j=0}^{nv} w_{ij}A_{ij}u^i v^i}{\sum_{i=0}^{nu}\sum_{j=0}^{nv} w_{ij}u^i v^i}$$

Where:

$nu = \text{'uorder'}-1$

$nv = \text{'vorder'}-1$

For the rational form the polynomial coefficients Aij are supplied:

$A_{00}, w_{00}, A_{10}, w_{10}, \ldots, A_{mo}, w_{m0}, A_{01}, w_{01}, \ldots, A_{m1}, w_{m1}, \ldots, A_{0n}, w_{0n}, \ldots, A_{mn}, w_{mn}$

starting with the constant term and ending with the term of the highest degree.

For the non-rational form the w's are missed out and the denominator is 1.

- Hermite coefficients SLBAHE

  This method can only be used for bicubics. The Hermite equation for the patch in matrix form is:

where M = (1 0 0 0)

(0 0 1 0)

$$P(u, v) = \frac{(1 u u^2 u^3) M A M^T (1 v v^2 v^3)^T}{(1 u u^2 u^3) M W M^T (1 v v^2 v^3)^T}$$

(-3 3 -2 -1)

(2 -2 1   1)

A = (w00*P00 w01*P01   wv00*Pv00   wv01*Pv01)

(w10*P10   w11*P11   wv10*Pv10   wv11*Pv11)

(wu00*Pu00 wu01*Pu01 wuv00*Puv00 wuv01*Puv01)

(wu10*Pu10 wu11*Pu11 wuv10*Puv10 wuv11*Puv11)

W = (w00 w01 wv00 wv01)

(w10 w11   wv10 wv11)

(wu00 wu01 wuv00 wuv01)

(wu10 wu11 wuv10 wuv11)

and the superscript T denotes the transpose.

In the matrices A and W, the coefficients P, Pu, Pv and Puv are the points at the corners and their derivatives. The w's are the corresponding weights and their derivatives. P00 denotes P(0,0), etc.

For the rational form the coefficients are supplied:

```
P00,    w00,    P10,     w10,     P01,    w01,    P11,     w11
Pu00,   wu00,   Pu10,   wu10,    Pu01,   wu01,   Pu11,   wu11
Pv00,    wv00,   Pv10,   wv10,   Pv01,   wv01,   Pv11,   wv11
Puv00, wuv00, Puv10, wuv10, Puv01, wuv01, Puv11, wuw11
```

For the non-rational form, the w's are missed out.

- Taylor series SLBATA:

  This method stores the derivatives evaluated

$$P(u, v) = \frac{\displaystyle\sum_{i=0}^{nu}\sum_{j=0}^{nv} \frac{w^{(i)(j)} P^{(i)(j)} u^i v^j}{i!j!}}{\displaystyle\sum_{i=0}^{nu}\sum_{j=0}^{nv} \frac{w^{(i)(j)} u^i v^j}{i!j!}}$$

Where:

$nu = \text{'uorder'}-1$

$nv = \text{'vorder'}-1$

$$P^{(i)(j)} = \frac{d^{i+j}P}{du^i dv^j}(0, 0)$$

$$w^{(i)(j)} = \frac{d^{i+j}w}{du^i dv^j}(0, 0)$$

The point is supplied first, followed by the u derivatives in order and ending with the derivative of order 'uorder'-1 in u, 'vorder'-1 in v.

CRRVSU - Create a surface of revolution

Receives

KI_tag_curve	*profil	curve to revolve
KI_vec_position	point	point on revolution axis
KI_vec_axis	direct	direction of revolution axis
<KI_int_nitems>	*nopts	number of options supplied
KI_cod_crop	opts[nopts]	array of options
<KI_tag_list>	optdta[nopts]	array of option data

Returns

KI_tag_surface	*revsur	resulting surface of revolution
KI_cod_error	*ifail	error code

Specific Errors

KI_impossible_swing	Cannot determine spun geometry
KI_su_self_intersect	Surface would be self-intersecting
KI_unsuitable_entity	Not one of the allowed curve types
KI_wrong_direction	Parameters in wrong order
KI_invalid_geometry	Invalid curve
KI_bad_parameter	End parameter out of range
	Start parameter out of range
KI_bad_parametric_prop	Inappropriate property
KI_bad_option_data	Should be exactly 2 values, option data missing

Description This function creates a spun surface or surface of revolution.

The curve 'profil' is revolved about the axis defined by the point vector 'point' and the direction vector 'direct', the new surface being the envelope of the curve. The sense of the revolution appears clockwise when viewed along the direction of the axis.

The curve must be one of the following:

- line
- circle
- ellipse
- B-curve

The options allow the caller to indicate whether simplification is to be attempted and to specify a parameter range which indicates which part of the curve to use. The possible entries are:

Token	Meaning	Data
CROPSI	Simplify to equivalent analytic surface	none
CROPPR	Parameter range on required section of `pro fil'	list of two doubles

The simplification token is CROPSI. If this is not given then the function will always return a surface of type TYSUSU. If it is given then, if possible, the function will return an simpler surface which is equivalent to the surface of revolution. Normally the simpler surface is analytic; for example, a spun line which intersects the axis is equivalent to a cone. Sometimes the simpler surface is a spun surface with a profile different to 'profil'. This can occur when the whole of 'profil' would sweep the spun surface twice, i.e. the surface would be self intersecting at all its points. For example, this occurs when 'profil' is a circle which lies in a plane perpendicular to the plane containing its centre and the spin axis. In this case, the profile of the resulting surface is a B-curve equivalent to the part of 'profil' specified using CROPPR (although the parameterisation of the new profile will, in general, be different to that of 'profil').

If the token CROPPR is supplied, then a list of two doubles must also be supplied as data. These two doubles specify a parameter range indicating the part of the curve to be spun. A parameter range may be specified for any case but will only be important in cases where the curve intersects the axis. In these cases, it is ambiguous as to which part of the curve should be spun and this option provides a way of choosing one part to be spun. The parameters must be given in ascending order. The curve is extended from the given parameter range until it meets the spin axis, and this becomes the portion of curve to be revolved. If ambiguous data is supplied and SLIPSI (see SEINTP) is non-zero, then an error is signalled, otherwise (if SLIPSI is zero) then the part of the curve used is undefined.

A self intersecting surface may be returned, but it is not possible to spin a line about an axis equivalent to the line itself.

If 'profil' is not an orphan, the resulting surface will be created as construction geometry in the owning part. A curve which is a dependent of another entity may be revolved - for example, a single curve may be the underlying curve of two spun surfaces.

CRSEPS - Sweep a B-curve into a B-surface

Receives

```
KI_tag_b_curve            *bc       B-curve to sweep
KI_vec_displacement        path     translation vector
```

Returns

```
KI_tag_b_surface          *bs       result of sweep
KI_cod_error              *ifail    failure code
```

Description The B-curve is moved along the path vector sweeping out a B-surface.

Returned surface 'bs':

- The curve lies along the v = 0 parameter line of the surface.
- The surface has the same dimension as the curve.
- The u order of the surface is the order of the curve.
- The v order of the surface is 2.
- The surface normal is in the direction of the cross product of the curve tangent and the path vector.

In general this procedure may construct a self-intersecting surface.

CRSHFA - Create sheet body from faces

Receives

```
KI_tag_list_faces        *faces    faces to use in sheet
```

Returns

```
KI_tag_body              *sheet    new body
KI_cod_error             *ifail    failure code
```

Specific Errors

```
KI_not_in_same_part  Faces not all from same body
KI_missing_geom      Face does not have surface
KI_non_manifold      Faces have non-manifold or disconnected boundary
KI_general_body      general body
```

Description The geometry, topology and associated data of the faces is copied into a new sheet body. The given faces must belong to the same body and must form a single connected set whose boundaries are manifold. Every face must have a surface attached.

This function is not supported for faces on general bodies.

CRSIPS - Swing a B-curve into a B-surface

Receives

```
KI_tag_b_curve      *bc       B-curve to swing
KI_vec_position      point    point on axis of rotation
KI_vec_axis          direct   direction of axis of rotation
KI_dbl_angle        *angle    angle of swing (in radians)
```

Returns

```
KI_tag_b_surface    *bs       result of swing
KI_cod_error        *ifail    failure code
```

Specific Errors

```
KI_bad_angle 'angle' must not be zero,'angle' must be
             between -2pi and 2pi
```

Description The B-curve is moved along an arc specified by the axis leaving a B-surface in its wake.

- 'angle' must not be greater than 2pi or less than -2pi.
- 'angle' must not be zero.

Returned surface 'bs':

- The curve lies along the v = 0 parameter line of the surface.
- The surface is rational.
- The u order of the surface is the order of the curve.
- The v order of the surface is 4.
- The surface will be periodic in u if the curve is periodic.
- The surface will be periodic in v if 'angle' is 2pi or -2pi.
- For positive angles the curve will be spun anticlockwise when viewed down the axis (i.e. the right hand screw rule applied to the axis gives the spin direction) and the surface normal will be in the direction of the cross product of the curve tangent and the spin direction.

In general this procedure may construct a self-intersecting surface.

CRSOFA - Create solid from faces

Receives

```
KI_tag_list_face    *faces   face(s) to copy into new body(s)
KI_cod_sllo         *action  type of action to mend wounds
                              SLLOCP => cap
                              SLLOGR => grow
                              SLLOGP => grow from parent
                              SLLORB => leave rubber
```

Returns

```
KI_tag_list_body    *bodys   new body(s)
KI_int_nitems       *nbodys  number of new bodies
KI_tag_list_int     *sbodys  state(s) of the body(s
                              RTLOOK => Valid
                              RTLONG => Negated
                              RTLOSX => Self-Intersecting
KI_cod_error        *ifail   failure code
```

Specific Errors

```
KI_wire_body         Unable to make solid from wire body
KI_dont_make_solid   Unable to make solid from face
KI_non_manifold      Can't heal wound with non-manifold boundary
KI_not_in_same_part  Faces not all from the same body
KI_general_body      General body unsuitable for sweep
```

Description The faces, which must all belong to the same body, are copied to make one or more new solids. The faces are NOT deleted from the body. A new body is made for each shell that has faces in the list. If all the faces of a shell are represented, a complete solid body is made; otherwise the missing faces of the shell are treated as holes to be healed. All holes in the new solids are covered by faces, the method used depending on the action code. All holes are covered using the same action. If the action is "cap", a new face is created for each hole and if possible a surface fitted which fits all the edges of the hole. If the action is "grow" the faces around each hole are extended (if possible) until they completely cover the hole. If the action is "grow from parent" new faces are created to cover the hole which are the inverse of extending the faces around 'faces' in the parent solid. If the action is "leave rubber" each hole is 'covered' by a rubber face.

Restrictions on growing:

- Edges of faces adjacent to a wound which do not form part of the loop of edges around the wound, but have a vertex on it, will not be allowed to contract back from that point. The "shrinkage" option implemented in DELFAS does not work for this routine.
- Each closed loop around a face or group of faces is healed independently. If the only solution would require more than one loop to be healed together it will not be found.
- The wound left by removing all the faces in a shell or body cannot be healed.

In general this procedure may give rise to self-intersecting object boundaries which could cause unpredictable errors later.

If local checking is on (see SEINTP and OUINTP), and the action is not "leave rubber", consistency checks will be made on newly created topological and geometrical entities, and the state of the body returned. A state of RTLOOK indicates the body is valid. A state of RTLONG indicates that the result body was originally "inside out" but has been negated, and is now "positive" (has positive volume) and valid. A state of RTLOSX indicates the body is self-intersecting and further modelling operations on it may fail. It is the responsibility of the calling routine to make any necessary reparation.

Note if rubber faces are included in the face list, they will be copied into the new solid. This solid could be made valid by further operations to replace the rubber faces. It is not possible to grow a rubber face to cover an adjacent wound.

If the session parameter for local checking is switched off or the action is leave rubber, the state returned will be RTLOOK regardless.

A self-intersecting body can be returned even if the ifail is zero.

This function is not supported for faces on general bodies.

CRSPCU - Create SP-curve(s) from B-spline data defined in surface parameter space

Receives

```
KI_tag_surface       *surf             basis surface for SP-curve
int                  *dim              dimension of control points
KI_int_order         *order            order of curve
KI_int_nitems        *nctrl            number of control points
KI_dbl_coefficients   ctrl[dim*nctrl]  control points
KI_dbl_knots          knots[]          knot vector
KI_cod_logical       *period           period flag
KI_cod_logical       *split            split flag
```

Returns

```
KI_int_nitems        *nspc             number of SP-curves returned
KI_tag_list_curve    *spc              SP-curves
KI_cod_error         *ifail            failure indicator
```

Specific Errors

KI_invalid_geometry	invalid SP-curves
KI_weight_le_0	weights are non-positive
KI_bad_knots	invalid knot vector
KI_bad_dimension	dimension must be 2 or 3
KI_linear_multi_seg	multi-segment linear curves not allowed
KI_insufficient_points	insufficient control points
KI_order_lt_2	order must be at least 2

Description Creates an SP-curve on a surface and B-spline data defined in the surface parameter space.

An SP-curve must be G1 continuous, and if periodic must meet itself with G1 continuity, an SP-curve may start or end on a surface degeneracy or singularity, surface degeneracies may only appear elsewhere on the curve if the entire curve lies within the degeneracy.

The arguments are:

Basis surface 'surf' of SP-curve.

- This is the surface in whose parameter space the curve is defined. The surface parameters are u and v in what follows.
- If 'surf' is construction geometry or is attached to a face, within a body, the SP-curve will be created as construction geometry in that body, and may only be attached to topology within that body.

Dimension of control points 'dim':

- For rational curves 'dim' = 3.
- For non-rational curves 'dim' = 2.

Order of the curve 'order':

- The order of the curve = degree + 1.
- The minimum order is 2.
- An order of 2 B-spline may consist of one segment only.

Number of control points 'nctrl':

- 'nctrl' >= 'order'.

Control points 'ctrl':

- For non-rational curves, the control points are points in the parameter space of 'surf'. They must be supplied as [u0,v0,u1,v1...].
- For rational curves each vector contains a point in parameter space followed by a weight for the point. The points are supplied [u0,v0,w0,u1,v1,w1...]. The weights must be positive.
- The (u,v) values defining control points do not have to lie within the parameter range defined by ENSUPA on limited surfaces, though ideally the B-spline curve so defined will be. If this is not the case, then an attempt will be made to extend the surface so that the curve lies wholly within it.
- The curve must be G1 continuous in parameter space, so that linear curves of >1 segment are not allowed.
- The curve will NOT be checked for self-intersection.

Knot vector 'knots':

- The knot values must form a non-decreasing sequence.
- The B-Spline may be closed, and may be periodic.
- For non-periodic B-Splines there must be ('nctrl' + 'order') knot values, the maximum multiplicity of an internal knot value is ('order' - 1), and the maximum multiplicity of an end knot value is 'order'.
- For periodic B-Splines there must be ('nctrl' + 1) knot values, the maximum multiplicity of any knot value is ('order' - 1). If the periodic knot has multiplicity greater than 1, repetitions must be given at the end of the knot vector.

Periodic flag 'period':

- The B-spline may be periodic if it is closed and meets itself with G1 continuity, and the resulting three space SP-curve is also closed and meets itself with G1 continuity.

Split flag 'split':

- If 'split' is set true an ordered list of SP-curves will be returned which satisfy continuity requirements. care should be taken that the degeneracy conditions are met. The flag may be turned off when it is known that the resulting single SP-curve is valid.

Ownership:

- The ownership of the newly created SP_Curve(s) is determined by the ownership of 'surf' as follows

`surf' owner	**SP_curve owner**
World	World
Assembly	Assembly
Body	Body
Face	Body owning face

Periodics:

If a single SP_Curve is returned, it will be periodic if the periodic flag 'period' was set KI_true. If 'period' was set KI_false but the SP_Curve is closed and G1 continuous, Parasolid may choose to treat the curve as periodic. OUSPCU is not affected and returns bspline data as received by CRSPCU, ENCUPA will however reflect this internal periodicity.

CRSPPC - Create B-curve by splining

Receives

```
KI_int_nitems       *npts           number of points supplied
KI_vec_position      pts[npts]      array of points to spline
<KI_int_nitems>     *nprops         number of curve properties
KI_cod_papr          props[nprops]  array of curve properties
<KI_tag_list>        pdata[nprops]  array of tags of data lists
```

Returns

```
KI_tag_b_paracurve  *bc             B-curve
KI_cod_error        *ifail          failure indicator
```

Specific Errors

KI_wrong_number_knots	wrong number of knots
KI_repeated_knots	repeated knots
KI_bad_knots	bad parameterisation
KI_bad_derivative	derivative too big, not enough coordinates in derivative
KI_bad_position	a spline point lies outside modeller size box
KI_coincident_points	repeated spline points
KI_insufficient_points	not enough spline points
KI_incompatible_props	periodic end condition with coincident end points, incompatible end conditions
KI_bad_parametric_prop	inappropriate property
KI_bad_tag_in_list	invalid null tag in pdata list

Description This function creates a B-curve by splining through a set of points. The curve will be continuous in slope and curvature. It may be periodic, in which case the end meets the start with slope and curvature continuity.

Number of points to spline 'npts':

- For non-periodic curves the minimum number of points is 2.
- For periodic curves the minimum number of points is 3.

Points to spline 'pts':

- Consecutive points should not coincide.
- To make a closed non-periodic curve the start point must be repeated at the end.
- If the curve is periodic the start and end points should not coincide.

Curve properties 'nprops', 'props', 'pdata'

There are several controls that may be applied to the splining operation. For example, the curve may be periodic or a knot vector may be supplied. Each action has a default, and each default can be overridden by giving a token in the 'props' array. 'nprops' is the number of tokens that have been supplied in 'props'.

A particular action may require additional data; if so, this must be supplied in a list. The tag of the list must be entered in the array 'pdata', in the position corresponding to the token in 'props'. If the action does not require additional data, then the null tag should be entered in the appropriate position in 'pdata'.

Property tokens:

The 'props' array contains 'nprops' tokens from the sequence PAPR00. The table shows which tokens may be used, and the data associated with them.

There are some pairs (or sets) of tokens which are alternatives; if both are supplied they may be contradictory, and in this case the last one to be supplied is the one which is used.

There are also cases in which the presence of a token implies a particular structure, and another implies a different structure. Use of both tokens is inconsistent, and raises an error.

Token	Meaning	Real Data
PAPRPE	curve is periodic	none
PAPRNS	no curvature at start of curve i.e. natural end condition	none

PAPRNE	no curvature at end of curve i.e. natural end condition	none
PAPRCS	derivative supplied at start i.e. clamped end condition	derivative vector
PAPRCE	derivative supplied at end i.e. clamped end condition	derivative vector
PAPRKT	knot vector supplied	knot vector
PAPRCU	force cubic curve	none

End conditions PAPRPE, PAPRNS, PAPRNE, PAPRCS, PAPRCE:

There are three end conditions available to control the splining: natural, clamped and periodic. Natural and clamped conditions refer to either the start or end of the splined curve, whereas the periodic end condition refers to both.

- Natural end conditions imply that the curve has no curvature at either the start or end of the curve. Natural end conditions are the default.
- Clamped end conditions allow the user to specify derivatives at either the start or the end of the curve, or both. Each derivative is supplied in a real list of length three.

  The derivatives should be supplied with respect to a parameter which varies from 0 to 1 between the first or last two points (as appropriate). In other words, the derivatives should have dimensions of length. The magnitude is significant, and supplying vectors which are too large may cause the curve to loop or kink.
- Periodic end conditions imply that the curve is closed, so that the curve will return to the start point after the final point has been splined. The curve will meet itself with continuity of tangent and curvature.

  If periodic end conditions are used, then at least three points must be supplied ('npts' >= 3).

Knot vector PAPRKT:

If a knot vector is supplied then it must satisfy the following conditions:

- The knot values must form an increasing sequence; repeated knots are not permitted.
- For non-periodic splining there must be 'npts' knot values.
- For periodic splining there must be ('npts'+1) knot values.

If the knot vector is not supplied then an accumulated chord length parametrisation is used.

Curve degree PAPRCU:

In general, the curve will be cubic. However, if only two points are given and the end conditions are natural, then a straight line (degree 1) will be produced by default. This default can be overridden by supplying the token PAPRCU, which forces the curve to be cubic.

CRSPPS - Create B-surface by splining

Receives

```
KI_int_nitems      *ncol              number of columns of points
KI_int_nitems      *nrow              number of rows of points
KI_vec_position     pts[ncol * nrow]
```

```
                                              mesh of points to spline
<KI_int_nitems>    *nprops                    number of surface properties
KI_cod_papr         props[nprops]             array of surface properties
<KI_tag_list>       pdata[nprops]             array of tags of data lists
```

Returns

```
KI_tag_b_parasurf  *bs                        B-surface
KI_cod_error       *ifail                     failure indicator
```

Specific Errors

KI_incompatible_props	periodic row/col with coincident end points, incompatible boundary conditions
KI_wrong_number_knots	wrong number of knots
KI_repeated_knots	repeated knots
KI_bad_knots	bad parameterisation
KI_bad_derivative	derivative too big, wrong number of coordinates in twist vector, wrong number of coordinates in derivative
KI_bad_position	a mesh point lies outside modeller size box
KI_coincident_points	repeated mesh points
KI_insufficient_points	not enough mesh points
KI_bad_parametric_prop	inappropriate property
KI_bad_tag_in_list	col knot vector tag is null, row knot vector tag is null

Description This function creates a B-surface by splining through a mesh of points. The surface will be continuous in slope and curvature. The points will lie on the surface, rows of points will define lines of constant v parameter, and columns of points will define lines of constant u parameter.

The surface may be periodic in either the u or v direction, in which case the surface 'wraps round' and meets itself along the relevant boundary, with slope and curvature continuity.

Number of columns of points 'ncol':

- For surfaces with non-periodic rows 'ncol' >= 2.
- For surfaces with periodic rows 'ncol' >= 3.

Number of rows of points 'nrow':

- For surfaces with non-periodic columns 'nrow' >= 2.
- For surfaces with periodic columns 'nrow' >= 3.

Mesh Points 'pts':

- The points must be supplied in order, row by row.
- Consecutive points, in a row or column, must not coincide.
- To make a surface closed and non-periodic along the rows the first and last points of each row must coincide.
- To make a surface closed and non-periodic along the columns the first and last points of each column must coincide.
- If the surface is periodic in u or v the first and last points of the rows or columns should not coincide.

Surface properties 'nprops', 'props', 'pdata':

There are several controls that may be applied to the splining operation. For example, the surface may be periodic or knot vectors may be supplied. Each action has a default, and each default can be overridden by giving a token in the 'props' array. 'nprops' is the number of tokens that have been supplied in 'props'.

A particular action may require additional data; if so, this must be supplied in a list. The tag of the list must be entered in the array 'pdata', in the position corresponding to the token in 'props'. If the action does not require additional data, then the null tag should be entered in the appropriate position in 'pdata'.

Property tokens:

The 'props' array contains 'nprops' tokens from the sequence PAPR00. The table shows which tokens may be used, and the data associated with them.

There are some pairs (or sets) of tokens which are alternatives; if both are supplied they may be contradictory, and in this case the last one to be supplied is the one which is used.

There are also cases in which the presence of a token implies a particular structure, and another implies a different structure. Use of both tokens is inconsistent, and raises an error.

In explaining the various controls that may be applied to the splining operation the following notation is used

- bottom boundary - first row of points
- top boundary - last row of points
- left boundary - first col of points
- right boundary - last col of points

Token	Meaning	Real Data
PAPRPU	surface rows are periodic	none
PAPRPV	surface columns are periodic	none
PAPRNB	natural boundary condition across bottom boundary of surface	none
PAPRNT	natural boundary condition across top boundary of surface	none
PAPRNL	natural boundary condition across left boundary of surface	none
PAPRNR	natural boundary condition across right boundary of surface	none
PAPRCB	derivatives supplied across bottom boundary of sur face i.e. clamped boundary conditions	`ncol' derivative vectors
PAPRCT	derivatives supplied across top boundary of surface i.e. clamped boundary condition	`ncol' derivative vectors
PAPRCL	derivatives supplied across left boundary of surface i.e. clamped boundary condition	`nrow' derivative vectors
PAPRCR	derivatives supplied across right boundary of surface i.e. clamped boundary condition	`nrow' derivative vectors

PAPRBL	bottom left twist vector supplied	twist vector
PAPRBR	bottom right twist vector supplied	twist vector
PAPRTL	top left twist vector supplied	twist vector
PAPRTR	top right twist vector supplied	twist vector
PAPRKU	knot vector supplied for rows	knot vector
PAPRKV	knot vector supplied for columns	knot vector
PAPRCU	force bicubic surface	none

End conditions - PAPRPU, PAPRPV, PAPRNB, PAPRNT, PAPRNL, PAPRNR, PAPRCB, PAPRCT, PAPRCL, PAPRCR:

There are three boundary conditions available to control the splining: natural, clamped and periodic. These apply to the u and v directions (i.e. the rows and columns) independently. Natural and clamped conditions refer to either the start or end of the rows or columns, whereas the periodic end condition refers to both the start and end. If no boundary condition is given for a boundary then natural end conditions are used. If clamped and natural boundary conditions are both supplied then only the last to be supplied is used; this is not flagged as an error.

- A natural boundary condition implies that the surface has no curvature across the relevant boundary. Natural end conditions are the default.
- A clamped boundary condition allows the user to specify derivatives across a boundary of the surface. The derivatives are supplied in a real list of length 3*'ncol' (for PAPRCB and PAPRCT), or 3*'nrow' (for PAPRCL and PAPRCR).
- The derivatives should be supplied with respect to a parameter which varies from 0 to 1 between the first or last two rows or columns of points (as appropriate). In other words, the derivatives should have dimensions of length. The magnitude is significant, and supplying vectors which are too large may cause the surface to loop or kink.
- A periodic boundary condition implies the surface is closed, so that the surface will return to the start row or column after the final row or column has been splined. The surface meets itself with continuity of tangent and curvature. If periodic end conditions are used, then at least three rows or columns must be supplied.

Twist vectors PAPRBL, PAPRBR, PAPRTL, PAPRTR:

A 'twist vector' is a derivative with respect to both u and v; i.e. it is the rate of change of the u derivatives in the v direction, and also the rate of change of the v derivatives in the u direction. The twist vectors may be supplied at any of the four corners, but only when both adjacent boundaries have clamped boundary conditions. If the twist vectors are supplied when they are not required then they are ignored. If the twist vectors are not supplied then a suitable default value is used.

Knot vectors PAPRKU, PAPRKV:

If a knot vector is supplied, in either direction, then it must satisfy the following conditions:

- The knot values must form an increasing sequence; repeated knots are not permitted.
- For non-periodic splining there must be 'ncol' knot values in the u direction, or 'nrow' knot values in the v direction.
- For periodic splining there must be ('ncol'+1) knot values in the u direction, or ('nrow'+1) knot values in the v direction.

If the knot vector is not supplied then an averaged accumulated chord length parametrisation is used.

Surface degree PAPRCU:

In general, the surface will be bicubic. However, if only two rows or columns of points are given and the corresponding end conditions are natural, then by default a ruled surface (degree 1) will be produced. This default can be overridden by supplying the token PAPRCU, which will force the surface to be bicubic. The surface is always non-rational.

CRSPSO - Create spherical solid

Receives

```
KI_vec_position          centre     center of sphere
KI_dbl_radius           *radius     radius of sphere
```

Returns

```
KI_tag_body             *sphere     sphere
KI_cod_error            *ifail      failure indicator
```

Specific Errors

```
KI_radius_too_large                Radius too large
```

Description A sphere is created of radius 'radius', with its center at 'centre'.

CRSPSU - Create a spherical surface

Receives

```
KI_vec_centre          centre     center position
KI_dbl_radius         *radius     radius of sphere
```

Returns

```
KI_tag_surface        *sphere     new spherical surface
KI_cod_error          *ifail      failure code
```

Specific Errors

```
KI_radius_too_large                radius too large
```

Description A new spherical surface is created with center 'centre' and radius 'radius'. The surface normal points away from the center position.

CRSPTC - Approximate a trimmed curve by an SP-curve

Receives

```
KI_tag_surface    *surf        surface upon which 't_cu' lies
KI_tag_curve      *t_cu        trimmed curve to approximate
double            *tol         required tolerance
KI_cod_logical    *degens      create degenerate SP-curves
KI_cod_logical    *sense       SP-curve sense
```

Returns

```
KI_int_nitems     *nspc        number of SP-curves returned
KI_tag_list_curve *spc         SP-curves
KI_cod_error      *ifail       failure indicator
```

Specific Errors

```
KI_failed_to_create_sp    failed to create SP-curve
KI_tolerance_too_tight    tolerance too tight
KI_bad_precision          tol less than modelling resolution
```

Description CRSPTC approximates 't_cu' which lies approximately on 'surf' by an SP-curve or series of SP-curves, lying on 'surf'.

The arguments are:

- Basis surface 'surf'.
  - Surface upon which to create SP-curve(s), 'surf' forms the basis surface of the new SP-curve(s).
  - If 'surf' is construction geometry or is attached to a face, within a body, the SP-curve will be created as construction geometry in that body, and may only be attached to topology within that body.
- Trimmed Curve 't_cu'.
  - Trimmed curve to be approximated by SP-curve(s)
  - Lies approximately on 'surf'. Note that if the trimmed curve is sufficiently long to overhang the ENSUPA range of a limited surface, then the surface may be extended in order to accomodate the SP-curve(s).
- Required tolerance 'tol'.
  - The SP-curve will be created to the distance tolerance 'tol', i.e. it lies within 'tol' of the image of 't_cu' in the surface. The image of 't_cu' in this context means the curve formed by the locus of points on 'surf' closest to corresponding points on 't_cu'.
  - If the required tolerance cannot be met then the function will fail with error KI_tolerance_too_tight indicating that a larger tolerance may allow a successful approximation.
- Production of zero length SP-curves 'degen'.
  - Sp-curves may start or end on surface degeneracies, may lie entirely within them, but may not pass through them. A trimmed curve which passes through a surface degeneracy will be approximated by at least two SP-curves. If 'degens' is true a zero length SP-curve will be returned which joins the otherwise disjoint

SP-curves in parameter space. Zero length SP-curves may not be attached to the model.

- SP-curve sense 'sense'
  - If 'sense' is true the created SP-curve will have the same sense as the trimmed curve in that it starts at the trimmed curve start. If 'sense' is false the SP-curve will start at the trimmed curve end and at end at the trimmed curve start.

The returns are:

- List of SP-curves 'spc' of length 'nspc'.

Ownership:

- The ownership of the newly created SP-curve(s) is determined by the ownership of 'surf' as follows

`surf' owner	SP_curve owner
World	World
Assembly	Assembly
Body	Body
Face	Body owning face

CRTOBY - Create the topology of a body

Receives

```
KI_cod_byty                 *b_type   body type
KI_tag_list_int             *types    topology types
KI_tag_list_int             *ids      integer id's of entities
KI_tag_list_<list>          *childs   children of entities
```

Returns

```
<KI_tag_list_<topology>>    *topols   tags of created entities
<KI_cod_rtto>               *retcod   fault found
<KI_int_id>                 *failid   id of faulty  entity
KI_cod_error                *ifail    failure code
```

Specific Errors

```
KI_bad_type               bad token in 'types'
KI_bad_value              identifier is zero or negative
KI_list_wrong_length      lists are not of same length
```

Description CRTOBY creates the topology of a body using the received data. A purely topological (rubber) body is created, which can then have geometry attached to it.

Received arguments: The body type 'b_type' is given by one of the tokens BYTYSO, BYTYSH, BYTYWR, and BYTYMN.

The three received lists are all of the same length, and have corresponding elements. The order of the lists does not matter, except that the first member of each list must refer to the body.

'types' is a list of tokens of topology types; the permitted values are TYTOBY, TYTOSH, TYTOFA, TYTOLO, TYTOED and TYTOVX. Token TYTOBY must occur just once, at the start of the list.

'ids' is a list of unique positive integer identifiers or "names" for all the topological entities in the body. The type of each identifier is given by the corresponding element in 'types'. Each id must be used just once. The id of the body will be the first element in the list.

'childs' is a list of lists of identifiers defining the connections between the topological entities in the body. Each list contains the integers (as defined in 'ids') denoting the entities directly belonging to the entity in the corresponding position in 'ids. Thus each list contained in 'childs' is a list of either the shells contained in the body, the faces in a shell, the loops in a face, the edges/vertex in a loop, or the vertices in an edge.

A special arrangement is made to indicate the direction or sense of edges in a loop: if the curve of an edge in a particular loop has a direction opposite to that of the loop, the id of the edge in the child list is negated.

The order of faces in a shell and of loops in a face does not matter, but the edges in a loop and the vertices in an edge must be correctly ordered, and the first shell in the list of children of the body must be the outer shell of the body.

If an entity has no children (e.g. any vertex) its children should be given as NULTAG.

The permitted types and numbers of children and parents for a solid body (type BYTYSO) are given as follows:

Entity	**Children**	**No. of Children**	**Parents**	**No. of Parents**
Body	Shells	>=1	-	-
Shell	Faces	>=1	Body	1
Face	Loops	>=0	Shell	1
Loop	Edges+senses, or single vertex	>=1	Face	1
Edge	Vertices	<=2	Loops	2 (counting senses)
Vertex	None	0	Edges/loop	>=1

This is the same for a sheet body (BYTYSH) except that the body must have just one shell.

In a wire body (BYTYWR), the body must have just one shell, and the shell must have just one face. If the wire is closed then the face must have two loops, if open just one. A vertex must have either one or two edges as parents. Otherwise the permitted numbers are as for a solid.

In a minimal body there are no edges; all other entities must have just one parent and one child, except for the vertex (with no children) and the body (with no parents).

Note: Each edge must be used twice as a child, once with each sense (this may be in the same loop).

A loop must have either a number of edges (with senses) as children, or a single vertex; it cannot have a mixture of edges and vertices, or more than one vertex.

A vertex must have a number of edges as parents, or a single loop; it cannot have both edges and loops, or more than one loop.

Returned arguments: If the received lists are valid a rubber body will be created; its tag, and those of the topological entities contained in it, will be given in 'topols'. 'topols' is of the same length as the input lists, and has corresponding elements: each tag in 'topols' refers to the same entity as the identifier at the same position in 'ids. So, in particular, the first tag in 'topols' will be that of the body.

Note that, although a face is required in the input data to construct the topology of a wire or minimal body, a face on such a body is now an illegal configuration. Therefore the face and its associated loop(s) are not created and 'topols' will contain NULTAG instead of the tags of the face and loop(s). In order to make a valid sheet body, any faces which will not have surfaces attached to them should be deleted using PIERCE.

If the input data is found to be invalid, an error code will be returned in 'retcod', and the id of the relevant input entity in 'failid'.

The following tokens may be returned in 'retcod':

Token	**Meaning**	**Corresponding failid**
RTTOOK	input is ok, valid topological body created	none
RTTOBB	bad body identifier - either it is not the first id, is not the only body id, or is not present at all	none
RTTODE	duplicate entry in 'ids': integer id occurs more than once	duplicated id
RTTOUC	undefined child: identifier in 'childs' is not in ids'; this in cludes negative ids not referring to edges	id of undefined child
RTTODC	duplicate child: entity has an id repeated in its child list	id of entity
RTTOWC	wrong type of children of entity	id of entity
RTTOFC	too few children of entity	id of entity
RTTOMC	too many children of entity	id of entity
RTTOWP	wrong type of parents of entity (see note below)	id of entity
RTTOFP	too few parents of entity	id of entity
RTTOMP	too many parents of entity	id of entity
RTTODW	disconnected wire: in a wire with two loops, the loops do not match	id of wire body
RTTOIL	invalid loop: the start and end vertices of the edges in a loop, taking sense into account, do not all match up	id of loop
RTTOCS	connected shells: two faces sharing an edge are in different shells	id of edge

RTTODS	disjoint shell: the face in a shell are not all connected (by edge)	id of shell
RTTONM	non-manifold vertex: this will be returned where the edges at a vertex are not all loop connected	id of vertex

A body will only be created if the input lists are found to be ok, i.e. if RTTOOK is returned in 'retcod'; otherwise, no topology will be created, and 'topols' will be returned as an empty list.

The lists will be checked for the above faults in the order given above, and the first one found will be returned. Thus, for instance, if RTTOMC is returned for an entity, none of the faults given by RTTONB, RTTODE, RTTOUC, RTTODC, RTTOWC, and RTTOFC will apply to the input lists.

fr

> **Note:** If an edge is used twice as a child with the same sense, RTTOMP is returned; if not used with both senses, RTTOFP is returned (if both occur RTTOFP takes priority).
>
> A loop with a mixture of edges and vertices as children will be returned with RTTOWC; a loop with more than one vertex as children will result in RTTOMC.
>
> A vertex with edges and loops as parents will result in RTTOWP; one with more than one loop parent will be returned with RTTOMP.
>
> A loop containing a ring edge (one with zero or one vertices) which also contains another edge will produce the return RTTOIL.

CRTOSO - Create toroidal solid

Receives

```
KI_vec_position           centre      centrerof torus
KI_vec_axis               axis        axis of torus
KI_dbl_radius            *majrad      major radius
KI_dbl_radius            *minrad      minor radius
```

Returns

```
KI_tag_body              *torus       torus
KI_cod_error             *ifail       failure indicator
```

Specific Errors

```
KI_majrad_minrad_mismatch Major radius not greater than minor radius
KI_radius_too_large       Minor radius too large, Major radius too large
```

Description A torus is created. Its size and position are equivalent to sweeping a circle radius 'minrad' about an axis through 'centre', direction 'axis'. The center of the (swept) circle traces a circular path (the spine circle) of radius 'majrad'.

CRTOSU - Create a toroidal surface

Receives

```
KI_vec_centre     centre          center position
KI_vec_axis       axis            axis direction
double           *majrad          annular radius
KI_dbl_radius    *minrad          radius of generating circle
```

Returns

```
KI_tag_surface   *torus           toroidal surface
KI_cod_error     *ifail           failure code
```

Specific Errors

```
KI_radius_sum_le_0    majrad < zero and majrad + minrad <= zero
KI_radius_too_large   invalid radius value
KI_radius_eq_0        invalid radius value
```

Description A complete toroidal surface is created with center 'centre', defined by rotating the generating circle (of radius 'minrad') about 'axis'. The major radius 'majrad' (radius of the spine circle) is the distance from 'centre' to the center of the generating circle.

The surface normal at any given position points away from the nearest point on the spine circle.

The locus of the torus is a closed surface sheet. However for certain choices of 'majrad' and 'minrad' this surface intersects itself creating two opposing singular points where the surface normal is undefined. The outer sheet of self intersection is called an apple. The inner sheet of self intersection is called a lemon. Each is bounded by the two singular points and treated as a separate surface.

This routine creates one of the three variations of a torus; non-self intersecting surface, lemon, or apple depending upon the values of 'majrad' and 'minrad'. If 'majrad' is positive and smaller than 'minrad' this defines an apple. If 'majrad' is negative (and 'majrad' plus 'minrad' is positive) this defines a lemon.

CRTRCU - Creates a trimmed curve

Receives

```
KI_tag_curve        *basis_curve      underlying basis curve
KI_dbl_parameter    *parm_1           start parameter
KI_dbl_parameter    *parm_2           end parameter
```

Returns

```
KI_tag_curve        *trimmed_curve    trimmed basis curve
KI_cod_error        *ifail            failure code
```

Specific Errors

```
KI_curve_too_short   trimmed curve is shorter than linear resolution
KI_bad_parameter     parameter 'parm_2' is out of range, parameter
                     'parm_1' is out of range, parm_2 is less than
                     parm_1 on non-periodic
KI_unsuitable_entity b-curve is referenced
```

Description Creates a trimmed curve from a curve and two bound parameters defining points on the curve.

The arguments are:

- The curve 'basis_curve' of the trimmed curve:
  - If the basis curve is already owned by a part, then the trimmed curve will also be owned by that part.
  - If the curve supplied is already a trimmed curve then its start and end points will be changed to be 'parm_1' and 'parm_2' repectively. This allows re-trimming of trimmed curves. The trimmed curve returned will have the same tag as that supplied.
- Parameter 'parm_1':
  - The parameter of the start point of the trimmed curve.
- Parameter 'parm_2':
  - The parameter of the end point of the trimmed curve.

If the basis curve is not periodic, the parameters must be valid for the curve (i.e. in the range returned by ENCUPA) and the corresponding points must be more than linear resolution apart. Furthermore 'parm_1' must be less than 'parm_2'.

If the basis curve is periodic, 'parm_1' and 'parm_2' can be in any range. If the points corresponding to 'parm_1' and 'parm_2' are less than linear resolution apart, then a trimmed curve is made from the whole basis curve.

If the basis curve is closed but not periodic, then a trimmed curve is made from the whole basis curve if the points corresponding to param_1 and parm_2 are within linear resolution of the ends of the curve and within linear resolution of each other.

CRTSFA - Creates a sheet body given surface and trimmed SP-curve data

Receives

KI_tag_surface	*sf	surface of face
KI_cod_logical	*sense	sense of face
<KI_tag_list_list>	*curves	curves, one list for each loop of face
double	*etol	3-space distance tolerance, stored on each edge
double	*ftol	3-space distance tolerance stored on the face
<KI_int_nitems>	*nopts	number of checking options
<KI_cod_tsop>	chopts[nopts]	checking options

Returns

<KI_tag_body>	*body	sheet body created
<KI_tag_face>	*face	face of 'body'
KI_cod_rtts	*state	state code
KI_cod_error	*ifail	failure indicator

Specific Errors

`KI_bad_sharing`	`sf referenced from other than supplied SP-curves`
`KI_bad_basis_surf`	`SP-curve not on surface`
`KI_has_parent`	`underlying SP-curve is not orphan, trimmed curve is not orphan, surface is not orphan`
`KI_wrong_entity_in_list`	`underlying curve is not an SP-curve, list contained non trimmed curve`
`KI_duplicate_list_item`	`curve geometry duplicated in lists`
`KI_unsuitable_entity`	`surface does not pass checks`
`KI_bad_option_data`	`incorrect checking option data`
`KI_bad_tolerance`	`face tolerance is less than Parasolid tolerance, edge tolerance is less than Parasolid tolerance`

Description This function creates a sheet body with a single face, given surface geometry for the face and curve geometry for the edges. The topology of the resulting sheet will be inferred from the geometry.

CRTSFA is designed primarily for importing geometric data of lower precision than Parasolid default. The geometry supplied to the routine need not, therefore, conform to Parasolid's standard tolerances. The user is responsible for specifying the distance tolerance parameters, the minimum distance two points have to be apart to be regarded as distinct, which will be stored with the face and edges of the resulting sheet body.

'sf' will be the geometry attached to the single face of the body. It can be of any type recognised by Parasolid, and must be orphan. If 'sf' is a B-surface, it must be capable of passing the continuity checks imposed by CHCKEN.

'sense' is a logical indicating whether the resultant face normal is aligned with the surface normal (set to true) or anti-aligned (false).

'curves' is a list containing lists of trimmed curves (type TYCUTR) representing the edge geometry of the face. The basis curves of the supplied trimmed curves must always be SP-curves. Each sub-list of 'curves' will contain the SP-curves geometry representing a single loop. 'curves' is not allowed to be null, except in the case of a wholly closed sheet with no loops, e.g. whole sphere or torus. The trimmed curves will be attached to the fins of the edges of the created face. Note that each curve appearing in 'curves' must be orphan, and must reference an underlying SP-curve which is also orphan.

The curve directions should be such that the intended face lies on the left of the curve when viewed in a direction opposing the face normal. The curves should be given in the correct order, following each other around the loop such that the end of any given curve is within tolerance of the start of the next one. Minimal corrective action will be attempted for input data not meeting these requirements, though this cannot be guaranteed to produce valid loops. The curve ordering must be correct as no attempt to reorder the curve is made.

> **Note:** CRTSFA can accept any trimmed surface formats supported by OUTSFA, although it will not incorporate SP-curves representing surface degeneracies into the model, as these have zero 3-space length. Similarly, trimmed curves shorter than the requisite tolerance (in terms of chord length) will not appear in the model (trimmed curves whose underlying SP-curve is closed are excepted).
>
> Two coincident opposing curves must be supplied if a wire edge (corresponding, for example, to a seam on a periodic surface) is required. In addition, CRTSFA will always create closed loop topology, regardless of the actual geometric closure of the supplied loops of SP-curves.

All SP-curves satisfy the continuity requirements described under CRSPCU before introduction into a model by means of CRTSFA.

The points stored on the vertices of the sheet body will be computed by Parasolid, so the application has no need to supply them. The position of the vertex is deemed to be the centroid point of all the trimmed curve ends meeting at the vertex. A suitable tolerance for the vertex is also computed. It should be noted that because the vertex point and tolerance are flexible to some extent, it is not necessary that consecutive trimmed curves meet to within twice 'etol' at the vertex. As a safeguard against potentially incorrectly ordered data, however, there will be a state code indicator returned when any computed vertex tolerance exceeds 10 times 'etol'.

'etol' is a 3-space distance parameter, and will be stored with all of the edges of the resultant sheet. Refer to SETLEN for the meaning of this tolerance within a model. From an application point of view, the value of 'etol' will reflect the accuracy of the sending system's curve data, and must be greater than or equal to Parasolid's linear tolerance. SETLEN can be used subsequently to modify the tolerance of any edge of the resultant sheet.

'ftol' is again a 3-space distance parameter, and will be stored with the face of the created sheet. Refer to SETLEN for the meaning of this tolerance within a model. From the application point of view, the value of 'ftol' will reflect the accuracy of the sending system's surface data. It is required that 'ftol' be less than or at most equal to 'etol'.

There are three optional levels of checking available on the resulting sheet body. These can be employed by the application according to how much is known about the input data in terms of, for example, the consistency of the loops directions. Whichever option is used, CRTSFA will attempt to correct any inconsistencies found within the limits imposed by the checks. Note that each option is independent of the others.

Option TSOPWR. This option allows the identification of wire topologies (e.g. seam-lines) and will ensure that the correct topology is made in such cases.

Option TSOPSX. This option enables the application to pick up self-intersections within the sheet. Checks made are:

- All edges are tested pairwise with each other to detect points at which their fin curves intersect which are NOT model vertices. There is no possible corrective action that can be taken here. Such sheets will fail CHCKEN.

Option TSOPLC. This will check that the loops of the face are consistent. With this option turned ON:

- Checks are made to check whether:
  - The loops are correctly contained (i.e. all inner hole loops lie within the boundary or peripheral loop, where such can be determined.
  - The combination of loops is a valid one for the surface type (for example, a set of loops on a plane contains just one peripheral positive loop and the rest negative hole ones).
- If the above conditions are not met the minimal corrective action is attempted. In an attempt to create valid loops CRTSFA may reverse loops, and investigate singularities of 'sf' introducing them into the face as isolated vertices if necessary. This corrective action is only attempted if the TSOPLC option is turned ON, and there are no guarantees of a valid result being found.

The resulting sheet is returned in 'body'.

'face' is the single face to which the supplied surface geometry is attached.

'state' is an indicator of the validity of the sheet after the chosen checking options have been performed on it. It is not intended to give detailed information (use CHCKEN for this). The returns are:

RTTSOK. Indicates that either all selected checks passed, and any attempted corrections were successful, or no checks were requested.

RTTSFR. Face is redundant with respect to the supplied tolerances. This is because all the supplied curves were too short with respect to 'etol' for the production of a valid face.

RTTSCI. Indicates that no sense could be made of the input loops of curves. This means that they did not follow round in the loop direction as indicated earlier, and there may be vertex tolerances computed as a result which exceed 10 times 'etol'.

RTTSSX. Indicates that the resultant sheet is self-intersecting because edges meet at a place other than a model vertex.

RTTSLI. Indicates that the loop structure of the resulting sheet is in some way invalid (e.g. there is no peripheral loop on a planar surface), and attempts to correct the problem failed. Such sheets should NOT be subsequently passed to CRKNPA or KNITEN, or otherwise incorporated into solid bodies.

RTTSEO. Indicates that the edges of the resulting sheet are incorrectly ordered at at least one of its vertices. This can be caused by the input data failing to comply with the convention on wire edges, for example.

DEFCON - Makes a connection between two entities

Receives

```
KI_tag_topology          *parent      parent entity
KI_tag_entity            *child       dependent entity
```

Returns

```
KI_cod_error             *ifail       failure code
```

Specific Errors

```
KI_has_parent            'child' already has a parent
KI_bad_shared_dep        A dependent of 'child' is illegally shared
KI_bad_shared_entity     'child' is illegally shared
KI_wrong_entity          Argument of wrong type
KI_not_in_same_partition Parent and child are in different partitions
```

Description Makes a connection between two entities. The acceptable combinations of parent and child are:

Parent	Child
Body	Point, Curve, Surface, List
Assembly	Point, Curve, Surface, List

All other connections are either made automatically (e.g. attributes and features are created and attached by single routine calls), or more specific routines must be used. (e.g. ATTGEO - attach geometry to topology).

DEFCON will not copy a 'child', thus attachments are further restricted by constraints on the sharing of entities:

- For a geometric entity to be attached to a part neither the entity nor any of its dependents may be attached to another topological entity. Also neither the entity nor any of its dependents can be shared with another orphan entity.
- A list cannot be attached to more than one part.

> **Note:** Lists attached to parts will not be archived with them by SAVMOD.

DEHOSH - Deletes a list of holes from a sheet body

Receives

```
KI_tag_body          *sheet    Sheet body
KI_tag_list_loop     *loops    loops to be deleted
```

Returns

```
KI_cod_error         *ifail    failure code
```

Specific Errors

```
KI_unsuitable_loop   loop is of wrong type
KI_missing_geom      sheet has no surface attached
KI_not_sheet         body is not a sheet
```

Description Deletes the list of interior loops from a sheet body.

The supplied body, `sheet' must be a sheet body having at least one face to which a surface is attached.

'loops' is a list of interior loops of the given sheet body. These loops must each be an interior loop in a face of the sheet. Each loop also must follow edges that bound exactly one face, e.g. wires, or the boundaries of holes. Should any loop supplied not fit the above requirements, the ifail KI_unsuitable_loop will be returned, otherwise all the loops will be deleted from the sheet.

DELCON - Breaks the connection between two entities

Receives

```
KI_tag_topology          *parent      parent entity
KI_tag_entity            *child       dependent entity
```

Returns

```
KI_cod_error             *ifail       failure code
```

Specific Errors

```
KI_not_connected              no such connection
KI_wrong_entity               argument of wrong type
```

Description The 'child' entity is detached from the 'parent' entity. The acceptable combinations of parent and child are:

Parent	Child
Body	Point, Curve, Surface, List
Assembly	Point, Curve, Surface, List

All other detachments are either made automatically (e.g. attributes and features are deleted without being detached first), or more specific routines must be used. (e.g. DETGEO - detach geometry from topology).

DELENT - Delete entity

Receives

```
KI_tag_entity            *entity      entity to delete
```

Returns

```
KI_cod_error             *ifail       failure code
```

Specific Errors

```
KI_still_referenced        argument is still referenced
KI_wrong_entity            type of 'entity' is incorrect
```

Description Deletes an entity, if possible. The following types of entity may be deleted:

- Assembly
- Instance
- Body
- Transformation
- Surface
- Curve
- Point
- List
- Attribute
- Feature

Bodies and assemblies may not be deleted if they are still instanced: any instances must be deleted first.

Attributes, features, instances, points and transformations can always be deleted.

An orphan or construction curve or surface may not be deleted if it is a dependent in another geometric entity.

A surface attached to a face may be deleted if it is not a dependent in another geometric entity. Attempting to delete a surface which is attached to a face and is a dependent in another geometric entity does not delete the surface but severs the connection between the surface and face. Similar conditions apply to curves and edges. Note : deleting a surface of type TYSUOF (offset surface) also deletes the underlying surface.

When a body is deleted all the topology, geometry and associated data in it are also deleted.

When an assembly is deleted the construction geometry, instances and associated data in it are also deleted.

DELFAS - Delete faces from body

Receives

```
KI_tag_list_face *faces          face(s) to be deleted
<KI_int_nitems>  *nopts          number of actions in 'optdta'
KI_cod_sllo       optdta[nopts]  type of action to mend wounds
                                  SLLOCP => cap
                                  SLLOGR => grow only
                                  SLLORB => leave rubber
                                  SLLOLT => loops together
                                  SLLOLI => loops independent
```

Returns

```
KI_tag_list_body *bodys          remaining body(s)
KI_int_nitems    *nbodys         number of bodies
KI_tag_list_int  *sbodys         state(s) of the body(s)
                                  RTLOOK => Valid
                                  RTLONG => Negated
                                  RTLOSX => Self-Intersecting
KI_cod_error     *ifail          failure code
```

Specific Errors

```
KI_wire_body          Unable to delete faces from wire body
KI_all_faces_in_body  Cannot delete all faces in body
KI_cant_heal_wound    Can't heal wound - impossible geometry
KI_non_manifold       Cannot heal wound with non-manifold boundary
KI_not_in_same_part   Faces not all from the same body
KI_general_body       General body unsuitable for sweep
```

Description The faces, which must belong to the same body, are deleted from the body. The faces must not form the shell of a sheet body (in this case DELENT should be used) but may form an inner shell (i.e. void) of a solid, in which case that shell is deleted. Rubber faces may be deleted from the body but the edges and faces around the wound must have geometry attached.

Any holes left by deleting the faces are healed according to the action code. If the action is "cap" then for each hole a new face is created and a surface found which fits all the edges of the hole. If the action is "grow only" the faces around the hole are extended until the hole is covered. Edges of faces adjacent to a wound which do not form part of the loop of edges around the wound, but have a vertex on it, will not be allowed to contract back from that point. If the "shrinkage" option is selected, the faces around the hole will be allowed to contract if a solution which extends the faces is not found first. If the action is "leave rubber" each hole is 'covered' by a rubber face. One of the actions "cap, "grow only", "shrinkage" or "leave rubber" must be supplied.

If the option "loops independent" is selected then each closed loop around a face or group of faces is healed independently. The default is to "heal loops together". Healing loops independently can result in a fragmented body.

In general this procedure may give rise to self-intersecting object boundaries which could cause unpredictable errors later.

If local checking is on (see SEINTP and OUINTP), and the action is not leave rubber, consistency checks will be made on newly created topological and geometrical entities, and the state of the body returned. A state of RTLOOK indicates the body is valid. A state of RTLONG indicates that the result body was originally "inside out" but has been negated, and is now "positive" (has positive volume) and valid. A state of RTLOSX indicates the body is self-intersecting and further modelling operations on it may fail. It is the responsibility of the calling routine to make any necessary reparation.

If the session parameter for local checking is switched off or the action is "leave rubber", the state returned will be RTLOOK regardless.

A self-intersecting body can be returned even if the ifail is zero.

This function is not supported for faces on general bodies.

DELIST - Delete list

Receives

```
KI_tag_list            *list        list to delete
```

Returns

```
KI_cod_error           *ifail       failure code
```

Specific Errors

```
KI_still_referenced  argument is still referenced
```

Description Deletes a list.

Can be called from the GO.

DELIVL - Delete items from a list

Receives

```
KI_tag_list    *list   list from which to delete items
KI_int_index   *startx position in list at which to start deleting items
KI_int_nitems  *nvals  number of items to delete
```

Returns

```
KI_cod_error   *ifail  failure indicator
```

Specific Errors

```
KI_list_too_short 'startx' + 'nvals' - 1 is more than list length
```

Description 'nvals' are deleted from the given list, starting from the position 'startx'.

Can be called from the GO.

DELSEN - Deletes a single geometric entity

Receives

```
KI_tag_geometry          *gm        geometry to delete
```

Returns

```
KI_cod_error             *ifail     failure code
```

Specific Errors

```
KI_still_referenced      argument is still referenced
KI_is_attached           geometry is attached to topology
```

Description Deletes a single geometric entity.

A geometric entity is deleted, without deleting any of its dependents. For example, deleting a trimmed curve will leave the underlying curve intact.

A geometric entity cannot be deleted if it is a dependent in another geometric entity or if it is attached to topology.

DETGEO - Detach geometry from topology

Receives

```
KI_tag_topology       *topol    topology to detach from
```

Returns

```
KI_cod_error          *ifail    failure indicator
```

Specific Errors

```
KI_no_geometry                no geometry attached
KI_wrong_entity               illegal topology
```

Description If 'topol' has geometry attached to it, the geometric entity is detached. The detached geometric entity is not deleted by DETGEO.

The operations performed by DETGEO are:

- Detach a surface from a face
- Detach a curve from an edge
- Detach a curve from a fin
- Detach a point from a vertex
- Detach a transform from an instance

DLENFE - Delete entity from feature

Receives

```
KI_tag_feature        *featre     feature
KI_tag_entity         *entity     entity to be removed
```

Returns

```
KI_cod_error          *ifail      failure code
```

Specific Errors

```
KI_not_in_feat             entity not found in feature
KI_wrong_type_for_feat     'entity' does not match feature type
```

Description Removes the entity from the feature.

DLORPH - Delete orphans

Receives

```
KI_cod_ty    *entype type of orphans to be deleted, TYENGE or TYADLI
```

Returns

```
KI_cod_error *ifail  failure indicator
```

Specific Errors

```
KI_bad_type    Specified type-code cannot have orphans
```

Description Entities of the specified type, which are not attached to any part, are deleted. Geometry entities, type TYENGE, and list entities, type TYADLI, may have orphans which can be deleted by a call to this routine.

ENBXEN - Enquire box containing the specified entity

Receives

```
KI_tag_topology    *entity     entity whose box is required
```

Returns

```
KI_dbl_box         entbox[6]   box containing entity
KI_cod_error       *ifail      failure indicator
```

Specific Errors

```
KI_system_error
KI_empty_body       body has no faces, edges, or vertices
KI_empty_assy       assembly instances no bodies
KI_missing_geom     entity has missing point geometry
KI_wrong_entity     entity is of incorrect type
```

Description The entity must be an assembly, body, face or edge.

The coordinates of the opposite corners of a rectangular box with sides parallel to X, Y and Z axes are returned. The box will contain the specified entity and will usually be close to the minimum possible size, but this is not guaranteed.

Element zero of array 'entbox' will contain the minimum X coordinate of the box, element one the minimum Y, element two the minimum Z, element three the maximum X and so on. On failure, all elements of 'entbox' will be set to zero.

If edge or face geometry is missing the entity will still be boxed ignoring those edges and faces. If any point geometry is missing then the boxing will fail.

An attempt to box an assembly which instances no bodies will fail.

If a body is empty the ifail KI_empty_body will be returned.

ENBYTY - Enquire body type

Receives

```
KI_tag_body    *body    body
```

Returns

```
KI_cod_enby    *bdytyp  type-code of body, from range ENBY00
KI_cod_error   *ifail   failure indicator
```

Description The type of body is returned as one of the following codes:

Token	Type
ENBYSO	solid
ENBYSH	sheet
ENBYWR	wire
ENBYMN	minimum object
ENBYGN	general

Can be called from the GO.

ENCONT - Enquire containment of point

Receives

```
KI_tag_point             *point        point
KI_tag_entity            *entity       entity
```

Returns

```
KI_cod_encl              *enclos       enclosure code
KI_cod_error             *ifail        failure code
```

Specific Errors

```
KI_unsuitable_entity      can't do test on invalid entity
KI_wrong_entity           'entity' of wrong type
KI_general_body           general body
```

Description Determines whether the point lies inside, outside or on the boundary of the entity.

The point must be a geometric point, not a vertex. The entity must be a face, edge, shell, body or assembly. The entity must check, if it doesn't the result may be incorrect.

The enclosure code will be one of:

Code	Meaning
ENCLIN	inside
ENCLOU	outside
ENCLON	on (the limits of)

The meaning of the three possible returns is obvious in most cases, but note the following:

- For an edge, a point that is outside may lie on the curve but outside the limits of the edge, or may not lie on the curve at all. Similarly for a face and its surface.
- For a face with holes, the interior of a hole is outside the face and the boundary of a hole is part of the boundary of the face.
- For a shell, the inside is the portion of space that is full of material; i.e. if the shell is the exterior of a body, any point within the body is inside the shell, even if it lies in a

void of the body. With respect to a void (interior) shell, points within the void are outside the shell.

- For a solid body with voids, points within a void are outside the body and the boundaries of the interior shells are part of the boundary of the body.
- For a sheet body, points are inside if they lie in the faces of the body, and on if they lie on edges or vertices on the boundary of the sheet.
- For a wire body, points are inside if they lie on the wire, and on if they lie on a terminating vertex of the wire.
- For an assembly, a point is inside if it is inside one or more of the body occurrences in the assembly; it is on the boundary if it is inside no body occurrence but is on the boundary of one or more body occurrences; otherwise it is outside.

This function is not supported for general bodies, shells in general bodies, or assemblies which contain a general body.

ENCUPA - Enquire curve parametrisation

Receives

```
KI_tag_curve        *curve        Curve for enquiry
```

Returns

```
KI_dbl               range[2]     Parameter range
KI_cod_papr          bounds[2]    Types of bound
KI_tag_list_int     *props        Parametrisation properties
KI_int_nitems       *nprops       Number of properties
KI_cod_error        *ifail        Failure code
```

Specific Errors

```
KI_invalid_geometry               curve not supported
```

Description This function returns details about the parametrisation of a specific curve.

'range':

The parameter of any point on the curve (as returned by ENPAPC) will lie between 'range[0]' and 'range[1]'. 'range[0]' < 'range[1]' always.

If the range is infinite, then finite values will be returned, and these will be large enough to ensure that the portion of the curve inside the size box is contained within the bounds.

If the curve is of type TYCUSP (SP-curve), the range parameters will be the first and last knot values of the underlying B-spline.

'bounds':

The parametrisation may behave in various ways at the end of its range. The 'bounds' array returns two tokens, describing the behavior at the start and end of the range respectively. The tokens may take the following values:

Token	Meaning
PAPRIF	infinite
PAPRXT	extendable
PAPRNX	not extendable

PAPRPE	periodic
PAPRDP	periodic, but not continuously differentiable across the boundary

If either end of the range is classified as extendable, then the range of the parameter can be altered by subsequent modelling operations, and the classification of the bounds can also change. In this case, it is advisable to call ENCUPA again. For all other bound classifications, the range and bound classification remain unchanged by modelling operations.

The periodic classifications (PAPRPE, PAPRDP) apply to both ends of the range - the bounds will be of the same type in these cases. If the type is PAPRDP, then the parametrisation function may have a derivative which is discontinuous in magnitude across the range boundary.

If the bound classification is infinite or periodic (PAPRIF, PAPRPE or PAPRDP), then the KI function ENPOPC will work on values beyond the corresponding range bound. Otherwise (if the classification is PAPRXT or PAPRNX) ENPOPC will only work for values within the range, unless the curve is a b-curve.

'props', 'nprops':

One or more properties of the parametrisation will be returned, in a list 'props' of length 'nprops'. The list contains tokens, which can take the following values:

Token	**Meaning**	
PAPRPE	periodic	
PAPRCN	all derivatives continuous	exactly one of these two properties will be returned
PAPRDC	all derivatives not necessarily continuous	
PAPRLI	linear	
PAPRCI	circular	
PAPRBC	bounds coincident (`range[0]' and `range[1]' correspond to the same point on the curve	

The PAPRLI and PAPRCI properties are only returned for lines and circles respectively, and are included for compatibility with the surface parametrisation functions.

The PAPRPE property indicates a periodic parametrisation.

Can be called from the GO.

ENDFAT - Enquire the attribute type definition of an attribute

Receives

```
KI_tag_attribute            *attrib      attribute
```

Returns

```
KI_tag_attrib_def           *type        attribute type
KI_cod_error                *ifail       error code
```

Description Returns the attribute type definition of the given attribute.

Can be called from the GO.

ENDFNM - Enquiry for an attribute type definition from its name

Receives

```
KI_int_nitems          *namlen          length of 'name'
KI_chr_string           name[namlen]    name of attribute type
```

Returns

```
<KI_tag_attrib_def>    *type            attribute type
KI_cod_error           *ifail           error code
```

Description ENDFNM returns the tag of the attribute type definition with the given name, or the null tag if there is no such attribute type definition.

The received arguments are handled in exactly the same way as the corresponding arguments of CRATDF, except that there is no restriction on 'name'; supplying SDL/TYSA_COLOUR, for instance, will return the tag of the system-defined color attribute.

If only the token identifier of an attribute type is known, the string identifier may be deduced from it; see the chapter on attributes, the appendix on system attribute type definitions in the Functional Description manual..

Can be called from the GO.

ENDIPE - Enquire discontinuities on a B-curve or B-surface

Receives

```
KI_tag_geometry       *geom   free form geometry
KI_cod_padi           *disc   discontinuities to return
```

Returns

```
KI_int_nitems         *ndisc  number of discontinuities found
<KI_tag_list_int>     *uorv   list of discontinuity types
<KI_tag_list_dbl>     *param  list of parameters
KI_cod_error          *ifail  failure code
```

Specific Errors

```
KI_wrong_entity      'geom' is not a B-curve or B-surface
```

Description This function will return the discontinuities of a B-curve or B-surface.

If a surface is received, each element in `uorv' will contain either PAPRUP or PAPRVP, depending on whether the corresponding parameter in `param' is the parameter of a discontinuity in the constant `u' direction or the constant `v' direction.

If a curve is supplied `uorv' will be null.

The `disc' argument indicates the level of discontinuities that are to be returned. At present the function only returns G1 discontinuities.

The parameters will be returned in increasing parameter order in `param'. For surfaces, the parameters in the `u' direction will be returned before the parameters in the `v' direction.

If there are no discontinuities on the geometry then `ndisc' will be zero and both `uorv' and `param' will be null.

ENEDTY - Enquire edge type

Receives

```
KI_tag_edge    *edge    edge
```

Returns

```
KI_cod_ened    *edtype  type-code of edge, from range ENED00
KI_cod_error   *ifail   failure indicator
```

Specific Errors

```
KI_general_body                      general body
```

Description The given edge is classified according to whether it is:

- open, closed or a ring.

  An open edge has different vertices at either end. A closed edge has the same vertex at either end. A ring edge has no vertices.

- 'wire', 'biwire' or normal.

  A normal edge has different faces and loops on either side. A wire has the same face on both sides: it may however have different loops on either side and in this case is termed a 'biwire'.

The type of edge is returned as one of the following codes:

Token	Type
ENEDON	Open, normal edge
ENEDCN	Closed, normal edge
ENEDRN	Ring, normal edge
ENEDOW	Open, wire edge
ENEDCW	Closed, wire edge
ENEDOB	Open, biwire edge
ENEDCB	Closed, biwire edge
ENEDRB	Ring, biwire edge

This is not supported for edges of general bodies.

Can be called from the GO.

ENENTY - Enquire entity type

Receives

```
<KI_tag_entity>  *entity         entity
KI_int_nitems    *ltypes         maximum length of types array
```

Returns

```
KI_cod_ty         types[ltypes]  type-codes of entity
KI_int_nitems    *ntypes         number of type-codes returned
KI_cod_error     *ifail          failure indicator
```

Specific Errors

KI_buffer_overflow	More than 'ltypes' type-codes to return

Description The entity with the given tag is found and a list of type-codes is returned which together define the type of the entity.

The first code will be from the range TYEN00 and will indicate whether the entity is topological, geometric, or associated data. The following type-codes will be drawn from various other ranges, determined hierarchically (for example a geometric entity may yield a list of type values 'geometric', 'curve', 'elliptic': TYENGE, TYGECU, TYCUEL).

If a null tag is given TYEN00 is returned.

Can be called from the GO.

ENEQGE - Enquire whether two geometries are equivalent

Receives

KI_tag_geometry	*geom1	first geometric item
KI_tag_geometry	*geom2	second geometric item

Returns

KI_cod_logical	*same	result. KI_true if same
KI_cod_error	*ifail	failure indicator

Specific Errors

KI_wrong_sub_type	only surfaces curves and points valid
KI_invalid_geometry	curve fails to pass checks, surface fails to pass checks
KI_bad_type_combn	geom1 and 2 are of different types

Description ENEQGE will compare two points, two curves, or two surfaces. If they occupy the same position(s) in space, 'same' will return KI_true, otherwise 'same' will return KI_false.

All comparisons of length or position are made to within a fixed resolution (see OUMODP).

ENEQGE is not guaranteed to detect equivalence but it will never claim two entities are equivalent when they are not.

Any curve or surface must be capable of passing the checks imposed by CHCKEN. The self intersection check is only performed if the appropriate option is set (see SEINTP).

ENEXEN - Enquire extreme point of entity

Receives

KI_tag_topology	*entity	face or edge
KI_vec_direction	dir1	first direction
KI_vec_direction	dir2	second direction
KI_vec_direction	dir3	third direction

Returns

KI_vec_position	posn	position of extreme point
KI_tag_topology	*ext	face, edge or vertex on which extreme lies
KI_cod_error	*ifail	failure indicator

Specific Errors

KI_cant_find_extreme	failure
KI_missing_geom	entity has missing geometry
KI_wrong_entity	entity is not a face or an edge
KI_coplanar	directions are coplanar

Description The entity must be a face or an edge.

The extreme point on the face or edge which is furthest in the direction 'dir1' is returned, unless there is more than one such point, in which case 'dir2' and 'dir3' are used successively to reduce the number of extreme points to one.

If the extreme point lies on a vertex or an edge, this is returned in 'ext'. Otherwise the face is returned.

The entity must not have missing geometry.

The three directions must not be coplanar.

ENFAPR - Enquire whether a face is parametrically rectangular

Receives

KI_tag_face	*face	face

Returns

KI_cod_logical	*rectan	parametrically rectangular flag
KI_dbl	urange[2]	parameter range in u
KI_dbl	vrange[2]	parameter range in v
KI_cod_error	*ifail	failure code

Specific Errors

KI_no_geometry	entity does not have geometry attached
KI_wrong_entity	entity is not a face

Description Determines whether a face is parametrically rectangular.

If 'rectan' is returned as KI_true then 'urange' and 'vrange' will contain the parameter limits of the rectangular area of the surface in which the face is defined.

For each pair of parameter limits, the following rules apply:

- The first element will always be less than the second.
- Both elements will lie inside the parameter range, as given by ENSUPA, unless the corresponding parameter is periodic. In that case the first will lie in the range, and the difference between the two will not exceed the period.

If 'rectan' is returned as KI_false then all the elements of 'urange' and 'vrange' will be 0.0.

ENFAPR does not guarantee to detect that a face is parametrically rectangular but it will never claim a face is parametrically rectangular when it is not.

ENLOTY - Enquire loop type

Receives

KI_tag_loop	*loop	loop

Returns

```
KI_cod_enlo    *lptype  type-code of loop, from range ENLO00
KI_cod_error   *ifail   failure indicator
```

Description The type of loop is returned as one of the following codes:

Token	Type
ENLOHO	hole loop
ENLOPE	peripheral loop
ENLONA	not applicable

ENLONA is returned in cases where the distinction between peripheral loops and holes does not make sense, for example the end loops of a cylindrical or toroidal tube.

ENPAPC - Find parameter of point on curve

Receives

```
KI_tag_curve       *curve    curve
KI_vec_position     coords   coordinates of point on curve
```

Returns

```
KI_dbl             *t        parameter of 'coords'
KI_cod_error       *ifail    failure code
```

Specific Errors

```
KI_not_on_curve     the supplied point is not on the curve
KI_invalid_geometry the curve fails to pass checks curve not supported
```

Description Returns the parameter of the given point, provided that it lies on the given curve. The parameter will lie in the range given by ENCUPA.

The curve must be capable of passing the checks imposed by CHCKEN. The self intersection check is only performed if the appropriate option is set (see SEINTP).

Can be called from the GO.

ENPAPS - Find parameters of point on surface

Receives

```
KI_tag_surface     *surf    surface
KI_vec_position     coords  coordinates of point on surface
```

Returns

```
KI_dbl             *u       u parameter of 'coords'
KI_dbl             *v       v parameter of 'coords'
KI_cod_error       *ifail   failure code
```

Specific Errors

```
KI_not_on_surface   point is not on the surface, point lies outside
                    the valid parameter range
KI_invalid_geometry the surface fails to pass checks, surface
                    not supported
```

Description Returns the parameters of the given point, provided that it lies on the given surface. The parameters will lie in the range given in ENSUPA for this surface. At parametric singularities, such as sphere poles, the non-degenerate parameter is returned as the lowest value in its range. The function works for any surface type.

The surface must be capable of passing the checks imposed by CHCKEN. The self intersection check is only performed if the appropriate option is set (see SEINTP).

Can be called from the GO.

ENPBEN - Calculates the parametric box of the given entity

Receives

```
KI_tag_topology *entity    entity whose parametric box is required
```

Returns

```
KI_dbl           ulimit[2] u parametric limits
KI_dbl           vlimit[2] v parametric limits
KI_cod_error    *ifail     failure indicator
```

Specific Errors

```
KI_missing_geom   the given face has no associated surface
KI_no_geometry    entity does not have geometry attached
KI_wrong_entity   entity is not a face
```

Description The entity must be a face.

U and v parametric limits are returned in two arrays. Element zero of 'ulimit' will contain the lower u value and element one the upper u. Element zero of 'vlimit' will contain the lower v and element one the upper v.

The u and v limits will bound an area of the surface in which the face is defined. The surface parameter space is defined by ENSUPA in two arrays 'urange' and 'vrange'. The parameter box is defined

(1)

```
'ulimit[0]' < 'ulimit[1]' and
'vlimit[0]' < 'vlimit[1]'
```

(2)

```
'ulimit[1]' - 'ulimit[0]' <= 'urange[1]' - 'urange[0]'
'vlimit[1]' - 'vlimit[0]' <= 'vrange[1]' - 'vrange[0]'
```

Hence for surfaces with periodic parameters the parameter box can never be larger than the period of the corresponding surface parameter.

(3)

```
'urange[0]' <= 'ulimit[0]' < 'urange[1]'
'vrange[0]' <= 'vlimit[0]' < 'vrange[1]'
```

So for a face that 'straddles' the boundary of a periodic parameter, the upper parameter value will be greater than the upper parameter range.

An attempt to find the parametric box of a face with no surface attached or insufficient geometry will fail.

ENPIFA - Enquire if points in face

Receives

```
KI_tag_face            *face              face
<KI_int_nitems>        *nparms            number of parameter pairs
double                 params[2*nparms]   u v parameter pairs
<KI_int_nitems>        *npvecs            number of pvecs
double                 pvecs[3*npvecs]    pvecs
<KI_int_nitems>        *nopts             number of options
KI_cod_pfop            opts[nopts]        options
<KI_tag_list_entity>   optdata[nopts]     option data
```

Returns

```
KI_cod_encl            enclos             enclosure
KI_tag_topology        topol              sub topology
KI_cod_error           *ifail             failure code
```

Specific Errors

```
KI_unsuitable_entity   can't do test on invalid entity
KI_no_geometry         no geometry on edge; no geometry on face
KI_bad_option_data     bad option data
KI_duplicate_item      duplicate item in option data
KI_not_same_length     arrays not the same length
KI_nitems_le_0         arrays both length zero
```

Description Determines whether the points lie inside, outside or on the boundary of the face.

The points must be given as either an array of parameter pairs ('params'), an array of co-ordinates ('pvecs') or both where the parameter pairs correspond to the co-ordinates in the same order (i.e. the number of parameter pairs 'nparms' and the number of pvecs 'npvecs' must be either equal or either 'nparms' or 'npvecs' = 0). It is the user's responsibility to ensure that if both 'pvecs' and 'params' are given that they correspond, the function will not detect if they don't and in this case the results are not guaranteed.

The option accepted is:

Token	Meaning
PFOPLO	enquire on loops of interest

The option PFOPLO takes data of a list of one or more loops from the face and considers the containment with respect only to these loops, i.e. if a face has a periphery loop and several hole loops and the PFOPLO option is used with only the periphery loop, points inside the holes will be regarded as inside the face. It is up to the user to ensure that the loops bound a finite area for meaningful results to be returned. The function will not detect if this has happened.

For each point an enclosure code is returned to classify the point relative to the face. The enclosure codes will be one of:

Token	Meaning
ENCLIN	the point lies inside the boundary

ENCLOU	the point lies outside the boundary
ENCLON	the point lies `on' the boundary of the face (i.e. on an edge or vertex)

The interior of a hole is outside the face and the boundary of a hole is part of the boundary of the face.

If the enclosure code is ENCLON, the corresponding element in 'topol' will be the vertex or edge that the point lies on, otherwise this will be null.

ENPOGC - Enquire point on general curve

Receives

```
KI_tag_curve        *curve    curve enquired of
KI_vec_position      coords   position of interest on curve
```

Returns

```
KI_vec_normal        tangnt   tangent direction ratio
KI_vec_direction     prnorm   normal direction ratio
KI_vec_normal        binorm   binormal direction ratio
KI_dbl_curvature    *curvat   curvature
KI_cod_error        *ifail    failure code
```

Specific Errors

```
KI_at_terminator    coords at terminator of curve
KI_not_on_curve     coords not within resolution distance of curve
KI_invalid_geometry the curve fails to pass checks
```

Description Calculates the tangent, principal normal, binormal and curvature of the curve at the given point. The first three are returned as direction ratios (normalized vectors).

The application program should check the curvature, and ignore the normals if the curvature is very small.

An example where this would occur is a straight line, which of course has a curvature of zero. In this case the principal normal and binormal directions are not returned.

The curve must be capable of passing the checks imposed by CHCKEN. The self intersection check is only performed if the appropriate option is set (see SEINTP).

ENPOGS - Enquire point on general surface

Receives

```
KI_tag_surface      *surfac  surface being enquired of
KI_vec_position      coords  position of interest on surface
```

Returns

```
KI_vec_normal        normal  surface normal
KI_vec_direction     prdir1  principal direction1
KI_vec_direction     prdir2  principal direction2
KI_dbl_curvature    *prcur1  principal curvature1
KI_dbl_curvature    *prcur2  principal curvature2
KI_cod_error        *ifail   failure indicator
```

Specific Errors

KI_at_singularity	coords at singularity of surface
KI_not_on_surface	coords not within resolution distance of surface
KI_invalid_geometry	the surface fails to pass checks

Description Calculates the normal and principal directions, and the principal curvatures, of the surface at the given point. The normal and principal directions are returned as direction ratios (normalized vectors).

If the principal curvatures are equal, the principal directions are not significant.

The surface must be capable of passing the checks imposed by CHCKEN. The self intersection check is only performed if the appropriate option is set (see SEINTP).

ENPOPC - Evaluate point from curve parameter

Receives

KI_tag_curve	*curve	curve
KI_dbl	*t	parameter of required point
<KI_int_nitems>	*ndrvs	number of derivatives required

Returns

KI_vec_derivatives	p[ndrvs+1]	point and derivatives
KI_cod_error	*ifail	failure code

Specific Errors

KI_bad_parameter	parameter out of range
KI_too_many_derivatives	too many derivatives requested
KI_invalid_geometry	curve not supported

Description Calculates the Cartesian coordinates of the point at the given parameter value 't', and also the derivatives with respect to 't', up to order 'ndrvs'.

Except for B-curves, which can be evaluated out of the range, if ENCUPA indicates that this curve is not periodic or infinite, then the parameter 't' must be in the range specified in ENCUPA for this curve.

Where derivative discontinuities exist at a junction between segments, the derivatives just after the discontinuity are returned.

For all curve types other than B-curves only derivatives up to and including second order may be requested.

In the case of foreign geometry curves (type TYCUFG), the maximum number of derivatives that can be returned by this function is determined by the curve's external evaluator. The evaluator should, at least, be able to calculate up to second order derivatives.

In the case of constant-parameter-line curves (type TYCUCP), the maximum number of derivatives that can be returned by this function would depend on the evaluators for the underlying surface geometry.

Can be called from the GO.

ENPOPS - Evaluate point from surface parameters

Receives

KI_tag_surface	*surf	surface
KI_dbl	*u	u parameter of required point
KI_dbl	*v	v parameter of required point
<KI_int_nitems>	*nudrvs	number of derivs wrt u required
<KI_int_nitems>	*nvdrvs	number of derivs wrt v required
KI_cod_logical	*nreq	request for normal

Returns

KI_vec_derivatives	p[(nudrvs+1) * (nvdrvs+1)]	point and derivatives
KI_vec_normal	norm	surface normal
KI_cod_error	*ifail	failure

Specific Errors

KI_at_singularity	failure to evaluate normal at singularity
KI_bad_parameter	parameter out of range
KI_too_many_derivatives	too many derivatives requested
KI_invalid_geometry	surface not supported

Description Calculates the Cartesian coordinates of the point at the given parameter values 'u' and 'v', and also the derivatives with respect to 'u' and 'v', up to order 'nudrvs' and 'nvdrvs' respectively. It also optionally calculates the surface normal. This normal will be parallel to the cross product of the derivative with respect to 'u' and the derivative with respect to 'v'. However, if the surface sense returned by OUTSUR is false, the sign of the normal will be reversed. The function works for any type of surface.

Except for B-surfaces and their offsets, which can be evaluated out of range, if ENSUPA indicates that this surface is not periodic or infinite (in either u or v), then the corresponding parameter must be in the range specified in ENSUPA for this surface.

Where derivative discontinuities occur at patch boundaries, the derivatives just after the discontinuity are returned.

There is a limit on the number of derivatives which may be requested.

In the case of foreign geometry surfaces (type TYSUFG), the maximum number of derivatives that can be returned by this function is determined by the surface's external evaluator. The evaluator should, at least, be able to calculate up to second order derivatives.

The following table shows the derivatives that can be requested for other surface types.

		u derivs			
		0	1	2	>2
v derivs	0	A	A	B	D
	1	A	A	C	D
	2	B	C	E	D
	>2	D	D	D	D

A - All surface types.

B - All surface types except blend surfaces.

C - All surface types except blend and offset surfaces.

D - B-Surfaces only.

E - All surface types except blend surfaces.

ENPOPS can be called with nudrvs = 2 and nvdrvs = 2 for offset surfaces, but only zeroth, first and second derivatives will be calculated. The uncalculated third and fourth derivatives will be returned in the derivative array as zero vectors.

The return argument 'p' is a vector array returning the point and possibly derivatives. These are stored as follows:

The point on the surface is the first vector in 'p' (i.e. vector 0)

The i'th derivative (i <= 'nudrvs') wrt u and

the j'th derivative (j <= 'nvdrvs') wrt v is vector

(i + ('nudrvs'+1) * j) of 'p'.

The return argument 'norm' contains the surface normal. The normal is only returned if 'nreq' is true. A value is returned even at a singular point of the surface, unlike ENPOGS. Note that a singular point has an ambiguous parametrisation, and different parameter pairs may give different normals, for the same point.

Can be called from the GO.

ENSHTY - Enquire shell type

Receives

```
KI_tag_shell   *shell   shell
```

Returns

```
KI_cod_ensh    *shtype  type-code of shell, from range ENSH00
KI_cod_error   *ifail   failure indicator
```

Specific Errors

```
KI_unsuitable_entity   Not a manifold shell
KI_general_body        general body
```

Description The given shell is classified according to whether it bounds a solid or a void. Every solid body must contain exactly one solid shell, and may contain any number of void shells. Sheet and wire bodies must contain exactly one shell.

The type of shell is returned as one of the following codes:

Token	Type
ENSHSO	solid
ENSHSH	sheet
ENSHWR	wire
ENSHVO	void

This is not supported for shells of general bodies.

ENSUPA - Enquire surface parametrisation

Receives

```
KI_tag_surface     *surf        Surface for enquiry
```

Returns

```
KI_dbl              urange[2]   Parameter range in u
KI_dbl              vrange[2]   Parameter range in v
KI_cod_papr         ubound[2]   Types of bound for u
KI_cod_papr         vbound[2]   Types of bound for v
KI_tag_list_int    *uprops      u parametrisation properties
KI_int_nitems      *nuprop      Number of properties in u
KI_tag_list_int    *vprops      v parametrisation properties
KI_int_nitems      *nvprop      Number of properties in v
KI_cod_error       *ifail       Failure code
```

Specific Errors

```
KI_invalid_geometry            surface not supported
```

Description This function returns details about the parametrisation of a specific surface.

'urange', 'vrange':

The u parameter of any point on the surface (as returned by ENPAPS) will lie between 'urange[0]' and 'urange[1]'; the v parameter will lie between 'vrange[0]' and 'vrange[1]'.

'urange[0]' < 'urange[1]' and

'vrange[0]' < 'vrange[1]' always.

If the range is infinite, then finite values will be returned, and these will be large enough to ensure that the portion of the surface inside the size box is contained within the bounds.

'ubound', 'vbound':

The parametrisation may behave in various ways at the end of its ranges. The 'ubound' and 'vbound' arrays each return two tokens, describing the behavior at the start and end of the ranges respectively. The tokens may take the following values:

Token	Meaning
PAPRIF	infinite
PAPRXT	extendable
PAPRNX	not extendable
PAPRPE	periodic
PAPRDP	periodic, but not continuously differentiable across the boundary
PAPRDG	degenerate

If either end of the range is classified as extendable, then the range of the parameter can be altered by subsequent modelling operations, and the classification of the bounds can also change. In this case, it is advisable to call ENSUPA again. For all other bound classifications, the range and bound classification remain unchanged by modelling operations.

The periodic classifications (PAPRPE, PAPRDP) apply to both ends of the range - the bounds will be of the same type in these cases. If the type is PAPRDP, then the parametrisation function may have a derivative which is discontinuous in magnitude across the range boundary.

If the bound classification is infinite or periodic (PAPRIF, PAPRPE or PAPRDP), then the KI function ENPOPS will work on values beyond the corresponding range bound. Otherwise (if the classification is PAPRXT or PAPRNX) ENPOPS will only work for values within the range, (unless the surface is b-surface or an offset b-surface..

If the 'ubound[0]' classification is degenerate, for example, then the v parameter is degenerate when u='urange[0]'. In other words, the derivative of the parametrisation function with respect to v is zero whenever u='urange[0]', for all values of v, and the parameter curve corresponding to u='urange[0]' degenerates to a single point. Note that a parametrisation can only degenerate at the end of its range.

'uprops', 'vprops', 'nuprop', 'nvprop':

One or more properties of the parametrisation will be returned, for both u and v, in two lists: 'uprops' of length 'nuprop', and 'vprops' of length 'nvprop'. The lists contain tokens, which can take the following values:

<table>
<tr><th>Token</th><th colspan="2">Meaning</th></tr>
<tr><td>PAPRPE</td><td colspan="2">periodic</td></tr>
<tr><td>PAPRCN</td><td>all derivatives continuous</td><td rowspan="2">exactly one of these two properties will be returned</td></tr>
<tr><td>PAPRDC</td><td>all derivatives not necessarily continuous</td></tr>
<tr><td>PAPRLI</td><td colspan="2">linear</td></tr>
<tr><td>PAPRCI</td><td colspan="2">circular</td></tr>
<tr><td>PAPRBC</td><td colspan="2">bounds at the ends of the parameter range are coincident</td></tr>
</table>

The PAPRLI property indicates that the corresponding parameter is proportional to the distance along a straight line. The straight line corresponds to a constant value of the other parameter.

The PAPRCI property indicates that the corresponding parameter represents an angle around a circle. The circle corresponds to a constant value of the other parameter.

The PAPRPE property indicates a periodic parametrisation.

ENVETY - Enquire vertex type

Receives

```
KI_tag_vertex   *vertex  vertex
```

Returns

```
KI_cod_enve     *vetype  type-code of vertex, from range ENVE00
KI_cod_error    *ifail   failure indicator
```

Specific Errors

```
KI_general_body                      general body
```

Description The type of vertex is returned as one of the following codes:

Token	Type
ENVEIS	isolated vertex (no edges)
ENVESP	spur vertex (single edge, one end only)
ENVEWR	wire vertex (all edges are wires)
ENVENO	normal vertex

For a definition of 'wires' see ENEDTY.

This is not supported for vertices on general bodies.

FIXIDS - Fix identifiers in part

Receives

```
KI_tag_part            *part    part in which to fix identifiers
```

Returns

```
<KI_int_nitems>        *nfault  number of entries in lists
<KI_tag_list_entity>   *entys   entities with new identifiers
<KI_tag_list_int>      *oldids  previous identifiers
<KI_tag_list_int>      *newids  new identifiers
KI_cod_error           *ifail   error code
```

Description FIXIDS searches 'part' for entities with invalid or duplicate identifiers, and assigns new unique identifiers to all such entities it finds.

If there are no faults found, 'nfault' is zero, and no other information is returned; otherwise the lists 'entys', 'oldids' and 'newids' are each of length 'nfault'.

'entys' contains the tags of entities to which new identifiers have been assigned; the old and new identifiers are returned in corresponding entries in 'oldids' and 'newids' respectively.

See IDENID and OUIDEN, for further explanation about identifiers.

FNENFE - Find entity in feature

Receives

```
KI_tag_feature      *featre   feature
KI_tag_entity       *entity   entity to be looked for
```

Returns

```
KI_cod_logical      *found    true if entity is in feature
KI_cod_error        *ifail    failure code
```

Specific Errors

```
KI_wrong_type_for_feat   'entity' does not match feature type
```

Description Returns true if the entity is in the feature, false otherwise.

Can be called from the GO.

GETMND - Recover a faulty model

Receives

```
KI_int_nchars           *keylen       length of key
KI_chr_key               key[keylen]  key of part
<KI_int_nitems>         *nopts        number of options
KI_cod_mdop              opts[nopts]  option codes
```

Returns

```
KI_tag_part             *part         recovered part
<KI_tag_list_int>       *mend         tokens describing mends
<KI_tag_list_int>       *fault        tokens describing faults
<KI_tag_list_<entity>>  *mcomp        mended components
<KI_tag_list_<entity>>  *fcomp        faulty components
<KI_int_nitems>         *nmend        number of mends returned
<KI_int_nitems>         *nfault       number of faults returned
KI_cod_error            *ifail        failure indicator
```

Specific Errors

```
KI_FG_receive_failure    part contains unrecognised foreign geom
KI_withdrawn_surface     part contains a withdrawn blend surface
KI_mend_attempt_failure  mending attempt failure
KI_schema_corrupt        contents of schema file not as expected
KI_schema_access_error   error opening, closing or reading the
                         schema file
KI_wrong_format          receiving binary archive as text or
                         vice-versa
KI_wrong_version         part archived by incompatible version of
                         modeller
KI_corrupt_file          invalid file contents
KI_usfd_mismatch         archive has wrong user-field size
KI_keyed_part_mismatch   archived part not same type as part with
                         same key
KI_cyclic_assy           receiving part would create cyclic
                         reference
KI_attr_defn_mismatch    archived attribute definitions don't
                         match current
KI_already_loaded        part with key already loaded
KI_file_access_error     error reading or closing archive
KI_cant_open_file        error opening archive
KI_key_not_found         key not found in archive
KI_bad_key               key has invalid syntax
```

Description GETMND attempts to load into internal memory a part which exists in an archive and which is identified with the given key, but which has either failed to be loaded using GETMOD or which has failed to check once loaded.

The same conditions apply to GETMND as to GETMOD, with regard to the interface parameter settings, and with regard to unloaded and loaded parts with the same key. This is reflected in the specific errors which can be returned.

Parts retrieved using GETMND will normally have the same part status as parts retrieved using GETMOD, though parts which have had to be modified in a fundamental way to make them valid, in particular which have had to be negated, will have the status of modified parts.

A diversity of problems may determine that a part which was valid within one version of the modeler becomes problematic or invalid within a later version:

- Withdrawn geometry types may not be received into a modeler which does not recognize them.
- Different versions of the modeler may impose different constraints on permissible model parameters, i.e. the values of linear and angular precision, and permitted model size. Versions of Romulus and Parasolid before v3.0 permit the user to set linear and angular precision parameters, and accordingly the maximum model size, whereas v3.0 Parasolid requires conformity to modeler parameters fixed internally, and outside the control of the user.
- The checker may vary in strictness between different versions of the modeler. For example, Romulus models were permitted to have vertices, edges or faces passing outside the size box; in Parasolid the checking routines require all parts to be wholly contained in the size box.
- Differences in the ways different versions of the modeler describe intersections between geometries (e.g. between two surfaces, between a curve and a surface) may lead to differences in the ways the consistency of a model is determined. For example, the surfaces of two adjacent faces in one version of the modeler may meet in an edge with a particular curve geometry; in a different version of the modeler the two surfaces may be deemed to meet at a curve geometry significantly different from the original.

The first two problems, a) and b), result in models failing to be received into the modeler using GETMOD. The latter two problems, c) and d), result in models failing to check, the first problem resulting in the "Model data is corrupt" failure, the second problem resulting in the "Geometry inconsistent with topology" failure.

GETMND provides a route by which some of these problems may be resolved, the type of mending tasks to be attempted being under the control of the user. Control is provided in the form of the opts[] array. A specific code will be required for each kind of mending task to be performed. The order in which options are supplied is of no significance. If no options are provided, then GETMND will function in much the same way as a call to GETMOD followed by one or more calls to CHCKEN (applied to the part itself if it is a body, or to all the un-keyed bodies within an assembly if the part is an assembly).

The range of options are as follows:

Option Code	**Interpretation**
MDOPMD	Attempt to bring model geometry into line with current standards of model con sistency
MDOPRB	Supply rubber geometry to faces, edges and vertices affected by types of geome try not recognized by the current modeler
MDOPNG	Negate any inside-out bodies

For those parts affected by withdrawn geometry types, but for which no relevant mending operation has been supplied GETMND will return a failure, in much the same way as GETMOD, returning KI_withdrawn_surface.

Once mending tasks have been performed, the part is checked in detail: a body will be fully checked, whilst an assembly will have all its un-keyed bodies fully checked. Information about components affected by mends and components faulty after mending attempts will be reported back via the lists 'mcomp' and 'fcomp'; the interpretation of the mends and faults are provided in accompanying token lists.

If there are no mends, 'nmend' is zero; similarly if there are no faults, 'nfault' is zero. Otherwise a token describing a mend or fault is returned in the token lists 'mends' and 'faults', and tags of mended or faulty entities are returned in the entity lists 'mcomp' and 'fcomp'. An entity may occur in both lists: for example, a face formerly attached to a withdrawn type of geometry may be rubberized according to a mending request option, but the face still qualifies as a fault. The owning shells, bodies, referencing instances of a faulty or mended component are to be identified using IDCOEN.

The lists 'faults' and 'fcomp' will be of equal length: there will be 'nfaults' entries in each, the 'i-th' fault type corresponding to the 'i-th' entry in 'fcomp'. When no data is associated with a given fault type the null tag will be entered in the appropriate place of 'fcomp'. Similarly the lists 'mends' and 'mcomp' will be of equal length, with a similar one-one correspondence between mend types and associated data.

If there is more than one fault in the entity then the function does not guarantee to return all the faults. Note that consistency mending performs a merge on the received part (see MERGEN for details), and this could succeed in making an invalid body valid without detecting that any mends have taken place.

The following tables show the tokens that may be returned, and the data associated with them. Different categories of fault and mend are separated for convenience:

Faults which indicate potential problems with a model, which when not accompanied by additional faults serve simply to warn the user of the possibility of problems. The report that a body is negative, however, may or may not constitute a fault; only the caller can decide.

Warnings:

Token	Meaning	Associated tag
RTSTSZ	Size setting differences	null tag
RTSTBX	Part not wholly in size box	tag of an example of violating item (typically a vertex, edge or face)
RTSTNG	Body is inside out	tag of inside-out body

When mending tasks have been performed information about them will be returned. Unless non-warning faults are returned in the fault list the tokens and accompanying data described below are simply for information.

Mends performed:

Token	Meaning	Associated tag
RTSTMD	Consistency mending has taken place	tag of list of affected edges and vertices
RTSTWG	Withdrawn geometries now rubber ized	tag of list of faces, edges or vertices affected by withdrawn geometry
RTSTIO	Body was inside out (now made positive)	tag of previously inside-out body

The following table describes faults which are serious and should be regarded as fatal to the secure use of the part within the current version of Parasolid. Missing geometry may not be serious in the context of a request to rubberize topology affected by withdrawn geometry types, though the problem is certainly serious if no remedy for the missing geometry problem can be found. The communication about the failure of the modeler to carry out a check indicates that the model cannot be processed reliably in its present form; though it also indicates an internal problem in the body checker, which should be reported.

Serious faults:

Token	Meaning	Associated tag
RTSTCR	data structure corrupt	tag of list of examples of corrupt topological or geometric entities
RTSTIN	inconsistent geom. and topology	tag of list of problematic topological items (not necessarily exhaustive)
RTSTMG	missing geometry	tag of list of topological items lacking geome try
RTSTSX	self-intersecting topology	tag of a list of topological items suffering from self-intersection
RTSTCF	failure in carrying out check	null tag

GETMOD - Get archived model

Receives

```
KI_int_nchars        *keylen         length of key
KI_chr_key           key[keylen]     key of part
```

Returns

```
KI_tag_part          *part           received part
KI_cod_error         *ifail          failure indicator
```

Specific Errors

```
KI_bad_field_conversion  oversize data read
KI_applio_not_registered application i/o functions not registered
KI_file_read_corruption  corrupt data read, perhaps an NFS problem
KI_FG_receive_failure    part contains irretrievable foreign geometry
KI_withdrawn_surface     part contains a withdrawn blend surface
KI_schema_corrupt        contents of schema file not as expected
KI_schema_access_error   error opening, closing or reading the
                         schema file
KI_wrong_format          receiving binary archive as text or
                         vice-versa
KI_wrong_version         part archived by incompatible version of
                         modeller
KI_more_than_one_part    more than one part
KI_corrupt_file          invalid file contents
KI_usfd_mismatch         archive has wrong user-field size
KI_keyed_part_mismatch   archived part not same type as part with
                         same key
KI_cyclic_assy           receiving part would create cyclic
                         reference
```

KI_attr_defn_mismatch	archived attribute definitions don't match current
KI_size_mismatch	archived part created with different size settings
KI_already_loaded	part with key already loaded
KI_file_access_error	error reading or closing archive
KI_cant_open_file	error opening archive
KI_key_not_found	key not found in archive
KI_bad_key	key has invalid syntax

Description GETMOD loads into internal memory a part which exists in an archive and which is identified with the given key.

Whether parts are received in text or in binary depends on the value of the SLIPBT interface parameter (see SEINTP for details). Error KI_wrong_format will result if an attempt is made either to receive in text a part transmitted in binary, or to receive in binary a part transmitted in text.

Whether or not user fields are received from an archive depends on the value of the SLIPUF interface parameter (see SEINTP for details). If user fields are not received, the user field of every entity in the archive will be filled with zeros. If they are received, all user fields will be restored at their values when the archive was made. A possible reason for not receiving user fields is that the current user-field size (as set by STAMOD) is not the same as when the archive was created - attempting to receive such an archive with user fields would give error KI_usfd_mismatch.

If the key is invalid (in the context of the archive system in use) KI_bad_key will be returned in 'ifail'.

If a part with the key cannot be found in the archive KI_key_not_found will be returned in 'ifail'.

We define the true-sub-parts of a stored (ENSTST) part S as those anonymous (ENSTAN) sub-parts of S reachable from S without encountering other stored parts.

A stored part is received with all its true-sub-parts, and any references between these newly received parts and those already in memory are resolved.

If any part, P, which they reference is not loaded a part node representing P is attached to the world as unloaded (ENSTUN).

If the given key does not match the key of any other part in internal memory the part in the archive with that key is received and attached to the world. The state of the newly received part will be stored.

If the given key matches that of an unloaded part the unloaded part is replaced with the newly received part.

If the key matches that of a modified (ENSTMD) part it is received as normal and attached to the world. There are then two parts with the same key in the world, one stored and the other modified.

If the key matches that of a stored part KI_already_loaded will be returned in 'ifail'.

R, an anonymous true-sub-part of P may instance another part, S, which is not a true-sub-part of P. If S is not in memory a node representing S is attached to the world as an unloaded part.

If S is in memory then R is made to instance S except if S is modified in which case a new unloaded part with the same key as S is attached to the world and instanced by R. For example; if we have the following parts in the archive:

```
st-key1
 \
  st-key2
```

and load the part with key2 into memory, modify it and then load the part with key1, we would have the following parts in memory:

```
st-key1
 \
  md-key2  un-key2
```

If it is desired that the part with key1 actually instance the modified part with key2 REDINS can be used to redirect the instance; i.e. to achieve:

```
  md-key1
 /
md-key2  un-key2
```

For an assembly, GETMOD will only explicitly load the assembly, its instances, and its true-sub-parts; stored-sub-parts referenced by the assembly will be loaded implicitly as required by later modeler operations. This has two implications:

- Modeler operations other than GETMOD may result in part-retrieval errors.
- Changes to interface parameters subsequent to the call to GETMOD on the owning-assembly may affect later retrieval of sub-parts referenced by that assembly. This may lead to errors, because, for example, sub-parts are being received in text rather than binary.

Note: It is always possible to force-load all sub-parts of an assembly by calling GETMOD explicitly on each one.

It is possible that receiving a part would cause a cyclic instance to be created. For example; imagine part P with key "k1" instances unloaded part Q; when Q is received it is discovered that it instances a part with the key "k1"; if the receive was completed P would instance Q which would instance P. In such a case KI_cyclic_assy is returned in 'ifail'.

As shown above it is possible to receive a part, P, with the same key as another modified or unloaded part, Q, already in memory. If after receiving P the modeler discovers that P and Q are not both assemblies or both bodies KI_keyed_part_mismatch is returned in 'ifail'.

When a part is created the modeler stores in the part the current values for the linear and angular resolution of the modeler and these values will be saved with the part in an archive. If, whilst receiving a saved part, the modeler discovers that the resolution values saved with the part are different from the current values, KI_size_mismatch will be returned in 'ifail'.

The definitions of any attributes attached to a part are stored with a part in the archives. When a part is received the modeler may discover a mismatch between an existing attribute definition and the stored definition for a particular attribute. If this occurs KI_attr_defn_mismatch will be returned in 'ifail'.

If the modeler discovers the contents of the archive to be corrupt KI_corrupt_file will be returned in 'ifail'.

Normally the parts in the archive will have been created with the same version of the modeler. However the modeler may discover a part created by a different version. In most cases the modeler will be able to cope and will receive the part without error. However if the version of the archived part is newer or much older than the current version the modeler will be unable to receive it and KI_wrong_version will be returned in 'ifail'.

If the part contains foreign geometry for which an evaluator is not available, the offending surfaces or curves are identified by having a system attribute of the type SDL/TYSA_BAD_FG attached. Bodies containing such geometry must not be used. This mechanism allows individual bodies in an assembly to be used despite the pressence of other bodies in the assembly containing unusable foreign geometry. Applications using foreign geometry must check for the presence of this attribute and delete such bodies.

If the transmit file contains more than one part (and so must have been transmitted by PK_PART_transmit), the ifail KI_more_than_one_part will be returned.

If, for any other reason, the modeler fails to successfully retrieve the part from the archive, KI_receive_failed will be returned in 'ifail'.

GETSNP - Restore a snapshot

Receives

```
KI_int_nchars    *nchars          number of characters in filename
KI_chr_filename   filnam[nchars]  filename for snapshot
int              *histfl          unused: should be zero
KI_cod_logical   *statfl          true to restore interface params
```

Returns

```
KI_cod_error     *ifail           error code
```

Specific Errors

```
KI_file_read_corruption  corrupt data read, perhaps an NFS problem
KI_FG_snapshot_failure   foreign geometry fails re-initialisation
KI_schema_corrupt        contents of schema file not as expected
KI_schema_access_error   error opening, closing or reading from
                         schema file
KI_wrong_version         snapshot from different version of modeller
KI_corrupt_file          invalid file contents
KI_usfd_mismatch         mismatch in user field size
KI_wrong_format          snapshot file in wrong format (binary or
                         text)
KI_file_access_error     error reading or closing snapshot file
KI_cant_open_file        cannot open snapshot file
KI_cant_find_file        cannot find snapshot file
KI_bad_filename          invalid filename
```

Description Restores tag memory from snapshot 'filnam'.

All entities which existed prior to the GETSNP call will be deleted if it is successful. After a successful call to GETSNP, there will be no session marks available, and each partition will have only its initial partition mark. In particular, it is not possible to roll back past a successful GETSNP.

If the GETSNP fails, it will be possible to roll back, except in the case of KI_FG_snapshot_failure.

Restored entities will have the same tags as when the snapshot was made. This means that the value of a restored tag may be the same as that of a tag allocated prior to the GETSNP call. To prevent confusion, the caller should ensure that all tags returned before the GETSNP call are discarded. The one exception is the tag of the 'world' entity which will be unchanged.

Interface and modelling parameters may be restored from the snapshot file, as follows. Some parameters are never restored; some are always restored; and the remainder are restored if 'statfl' is true, but retain their original value if 'statf' is false:

Not restored	SLIPCH, SLIPJO, SLIPRB, SLIPRF
Restored	SLIPBB, SLMPLP, SLMPAP
Restored if 'statfl' = true	SLIPBT, SLIPSW, SLIPLC, SLIPDC, SLIPUF, SLIPSI, SLIPCO, SLIPTL, SLIPGS, SLIPGT

GTINLI - Get values from a list of integers

Receives

```
KI_tag_list_int *list        list from which to extract items
KI_int_index    *startx      position in list from which first value
                             is to be extracted
KI_int_nitems   *nvals       number of values to extract from list
```

Returns

```
int              vals[nvals] values extracted from list
KI_cod_error    *ifail       failure indicator
```

Specific Errors

```
KI_list_too_short 'startx' + 'nvals' - 1 is more than list length
KI_bad_index      'startx' is not in range 1 to list length + 1
```

Description The given values are extracted from the list. The first value is taken from the 'startx' position, the second from 'startx' + 1 and so on. The first element in the list is number 1.

Can be called from the GO.

GTRLLI - Get values from a list of reals

Receives

```
KI_tag_list_dbl *list        list from which to extract items
KI_int_index    *startx      position in list from which first value is to be extracted
KI_int_nitems   *nvals       number of values to extract from list
```

Returns

```
double           rvals[nvals] values extracted from list
KI_cod_error    *ifail        failure indicator
```

Specific Errors

```
KI_list_too_short 'startx' + 'nvals' - 1 is more than list length
KI_bad_index      'startx' not in range 1 to list length + 1
```

Description The given values are extracted from the list. The first value is taken from the 'startx' position, the second from 'startx' + 1 and so on. The first element in the list is number 1.

Can be called from the GO.

GTTGLI - Get values from a list of tags

Receives

```
KI_tag_list_<tag> *list         list from which to extract tags
KI_int_index      *startx       position in list from which first is
                                to be extracted
KI_int_nitems     *ntags        number of tags to extra from list
```

Returns

```
KI_tag             tags[ntags]  values extracted from list
KI_cod_error      *ifail        failure indicator
```

Specific Errors

```
KI_list_too_short  'startx' + 'ntags' - 1 is more than list length
KI_bad_index       'startx' not in range 1 to list length + 1
```

Description The given values are extracted from the list. The first value is taken from the 'startx' position, the second from 'startx' + 1 and so on. The first element in the list is number 1.

GTTGLI does not do any checks on the validity of the tags extracted.

Can be called from the GO.

HOLLBY - Hollows a solid body

Receives

```
KI_tag_body           *body     body to be hollowed
KI_dbl                *offset   default offset
KI_cod_logical        *check    level of checking required
<KI_tag_list_face>    *pierce   faces not to be offset
<KI_tag_list_face>    *faces    faces offset by other amounts
<KI_tag_list_dbl>     *dists    list of other offset distances
KI_dbl_distance       *tol      maximum applied tolerance
<KI_int_nitems>       *maxflts  maximum number of entities in badtag
```

Returns

```
<KI_tag_list_entity> *oldfas   list of original faces offset
<KI_tag_list_entity> *newfas   list of corresponding new faces
<KI_tag_list_entity> *badtag   entities which caused problems
KI_cod_rtof          *state    state of body after hollow
KI_cod_error         *ifail    failure code
```

Specific Errors

```
KI_cant_hollow         failed to hollow body
KI_boolean_failure     boolean failed
KI_not_unique          boolean resulted in more than one body
KI_unsuitable_entity   could not duplicate body; body is not a solid
                       body
KI_bad_tolerance       proposed tolerance is too small
```

`KI_bad_value`	`non-default offset too small; default offset too small`
`KI_not_in_same_body`	`offset face is not in supplied body; pierced face is not in supplied body`
`KI_duplicate_list_item`	`face is in both pierced and faces`
`KI_list_wrong_length`	`list of faces and dists not same length; too many faces in lists pierced and faces; too many faces for non-default offset; too many pierced faces`
`KI_list_too_short`	`no dists supplied for faces`

Description The 'body' is hollowed. The 'offset' argument gives the default offset of the hollowing - the thickness of the skin of the resulting body. A positive 'offset' will offset the body outwards (i.e. in the direction of the face normals) and a negative value inwards.

Faces which are to be offset by amounts other than the default can be specified in the 'faces' argument with the specific offset for this face supplied in the corresponding position in the 'dists' list. Notice that using these arguments allows a 'mixed' hollow to be defined with some faces being offset inwards and some outwards.

Faces which are not to be offset at all are defined in the 'pierce' list. If no faces are pierced the result of hollowing a body will be to generate a body containing a void. Pierced faces remain only as a section through the hollowed body. For instance, hollowing a cylindrical solid with one end face pierced will result in a tube with one open and one closed end.

Under some circumstances the function may need to replace exact geometry by tolerant geometry. For instance, a four-edge vertex in general will offset to two three-edge vertices and a connecting edge. If this new edge is smaller than the supplied tolerance then the vertex becomes tolerant and no new edge is introduced. In all situations where approximation is required the new geometry will have a tolerance less than or equal to the tolerance supplied through the 'tol' argument.

Many circumstances can give rise to changes in topology . Amongst them are:

- Dealing with geometry which fails to offset. It is known that the offset surface of a face would be self-intersecting an attempt is made to either remove the offset face. For example, a blend may be removed from an edge. The investigation of self-intersection is not exhaustive, however, and it can occur that instances are not trapped.
- Dealing with configurations which can be repaired. For instance, an edge can offset to a point or a face can become absorbed into the body.

The extent to which checking is applied to the body is specified by the 'check' argument. If 'check' is true then face-face checks are done on the body in addition to default checks. For most applications setting 'check' false will give an adequate level of checking.

The tag of a face in the original body will remain on the exterior of the resulting body and will retain the same sense.

Each face to be offset will have a partner created in the final body which is an offset of the original face. The pairs of original and new faces are returned in the 'oldfas' and 'newfas' arguments. Both 'oldfas' and 'newfas' can contain null tags. This will occur when faces are removed for the reasons given above.

The error reporting scheme comprises the four arguments 'badtag', 'state', 'mxflts' and 'ifail'. A non-zero 'ifail' is reserved for reporting unsuitable arguments to the function, failures in the internal boolean operation and system errors.

Algorithmic failures where the items causing the failure can be identified result in a zero 'ifail' and more specific information being returned in the 'state' argument and 'badtag'. For example, if new geometry cannot be found for an edge 'state' RTOFEM will be returned and the tag of the edge whose curve could not be found in 'badtag'. The user may set an upper limit on the number of faults found in 'mxflts'; if 'mxflts' is zero then no limit is applied.

'state' refers to the validity of the item after modification and not to its original validity and after such a failure 'body' may be corrupt and a rollback should be performed. For this reason tags of new paired and unpaired topology cannot be returned in 'badtag' and the tags will refer to topology in the original body supplied. In the case of new unpaired edges and vertices, tags will refer to adjacent faces in the original body. For example if a new unpaired edge cannot be modified a 'state' RTOFEM is returned and 'badtag' and will contain the tag of a list of the two faces whose offsets are to be used to find the new curve. 'badtag' is a list of items which failed for the reason indicated in the 'state' argument.

Possible values of 'state' and the contents of elements of 'badtag' are :

Token	**Tag on badtag list**	**Meaning**
RTOFOK	null	Body is OK
RTOFSO	face	Surface failed to offset or face could not be deleted
RTOFVM	vertex list faces	Failed to find new geometry for vertex Failed to find geometry for new unpaired vertex
RTOFEM	edge list faces	Failed to find new geometry for edge Failed to find geometry for new unpaired edge
RTOFVT	edge list faces	Edge should have disappearde New unpaired edge should have disappeared
RTOFFA	face	Face failed checks
RTOFSX	list faces	Pair of faces where face-face inconsistency found

Notice that a successful execution of the hollowing operation is indicated by:

'ifail' returning KI_no_errors and 'state' returning RTOFOK

This function is not supported for general bodies.

IDATEN - Enquire the attribute of a given type attached to an entity

Receives

```
KI_tag_entity          *entity   entity in which to look
KI_tag_attrib_def      *type     type of attribute to look for
```

Returns

```
<KI_tag_attribute>     *attrib   attribute
KI_cod_error           *ifail    error code
```

Specific Errors

```
KI_wrong_entity   'entity' cannot own attributes of 'type'
```

Description Returns the attribute of 'type' attached to 'entity', or the null tag if there is no such attribute. If the given 'entity' is not permitted to own an attribute of the given 'type', KI_wrong_entity is returned in 'ifail'.

If IDATEN is used for attributes of class RQAC06 or RQAC07, it will only return one attribute, even if several are present.

Can be called from the GO.

IDATLS - Enquire the attributes of a given type attached to an entity

Receives

```
KI_tag_entity           *entity  entity in which to look
KI_tag_attrib_def       *type    type of attribute to look for
```

Returns

```
<KI_tag_list_attribute>  *atlist  list of attributes
KI_cod_error             *ifail   error code
```

Specific Errors

```
KI_wrong_entity   'entity' cannot own attributes of 'type'
```

Description Returns a list of all the attributes of 'type' attached to 'entity', or the null tag if there are no such attributes. If the given 'entity' is not permitted to own attributes of the given 'type', KI_wrong_entity is returned in 'ifail'.

Can be called from the GO.

IDATPA - Identify all the attributes of a given type in a part

Receives

```
KI_tag_part             *part    part in which to look
KI_tag_attrib_def       *type    type of attribute to look for
```

Returns

```
<KI_tag_list_attribute>  *atlist  list of attributes
KI_cod_error             *ifail   error code
```

Description Returns a list of all the attributes of the given type attached to entities in the given part, or the null tag if there are no such attributes.

Can be called from the GO.

IDCCEN - Identify common connected entities

Receives

```
KI_tag_entity          *enty1   first entity to look in
<KI_tag_entity>        *enty2   second entity to look in
KI_cod_idty            *idty    type of connection
```

Returns

```
KI_tag_list_entity     *entys    returned list of entities
<KI_int_nitems>        *nitems   number of returned entities
KI_cod_error           *ifail    failure indicator
```

Specific Errors

```
KI_general_body         general body
KI_not_in_same_body     entities not owned by the same body
KI_wrong_entity         wrong entity type(s)
```

Description Identify and return a list of entities connected to the received entity or entities and with the specified connection.

All connections are found via the topology of the part; e.g. to identify all curves common to two surfaces, all the faces associated with the first surface are compared with all the faces associated with the second surface to find all common edges and hence all common curves.

The supported connection types are:-

IDTYCS: Find common curves from two surfaces

Both surfaces must be owned by topology in the same part

enty1, enty2 = surfaces

entys = list of curves (May contain duplicate entries)

IDTYSC: Find pairs of surfaces given common curve

enty1 = curve

entys = list of pairs of surfaces

IDTYEF: Find edges common to two faces

Both faces must be owned by the same part

enty1, enty2 = faces

entys = list of edges

Can be called from the GO.

This function is not supported for entities belonging to general bodies.

IDCOEN - Identify connected entities

Receives

```
KI_tag_entity             *entity   entity to look in
KI_cod_ty                 *contyp   type of desired entities
```

Returns

```
KI_tag_list_<entity>      *entys    connected entities found
<KI_int_nitems>           *nentys   number found
KI_cod_error              *ifail    failure indicator
```

Specific Errors

```
KI_bad_type_combn              bad type combination
KI_general_body                general body
```

Description IDCOEN creates a list of the entities of type 'contyp' connected to 'entity'.

The input entity may not be from a general body. These enquiries for general bodies are supported in the PK interface.

The following table indicates the valid combinations of the type 'entity' and 'contyp':

	TYTO..									TYGE..				TYAD..				TYOWNR
	AS	IN	BY	SH	FA	LO	FN	ED	VX	SU	CU	PT	TF	AD	AT	LI	FE	
WO	n		n							n	n	n	n	n		n		
AS	n	n	n							n	n	n			n	n	n	n
IN	1*		1*										1*		n		n	1
BY	n	n		N	n	n	n	n	n	n	n	n			n	n	n	
SH			1		n	n	n	n	n						n			
FA			1	1		n	n	n	n	1*					n		n	
LO			1	1	1		n	n+	n+						n		n	
FN			1	1	1	1		1	2*		1*							
ED			1	1	2!	2!	2?		2*		1*				n		n	
VX			1	1	n	n	n	n+				1*			n		n	
SU	1*		1*		n										n		n	1*
CU	1*		1*				1*	n							n		n	1*
PT	1*		1*						1*						n		n	1*
TF		1*																
AT	1*	1*	1*	1*	1*	1*		1*	1*	1*	1*	1*					1*	1
LI	1*		1*															1*
FE	1*	n	1*		n	n		n	n	n	n	n			n			1
	AS	IN	BY	SH	FA	LO	FN	ED	VX	SU	CU	PT	TF	AD	AT	LI	FE	
	TYTO..									TYGE..				TYAD..				TYOWNR

The blank cells indicate an invalid combination and the other symbols indicate how many connected entities may be returned as follows:

1 => Exactly one

1* => Zero or one

2 => Exactly two

2- => One or two

2* => Zero, one or two

2! => Exactly two; duplicates possible; either or both may be NULTAG

2? => Exactly two; no dupliactes except either or both may be NULTAG n => Zero, one or more

n+ => Zero, one or more; duplicates possible

N => One or more

When the loops or faces of an edge are sought, the left hand loop or face is returned first. Left and right are determined with respect to the direction of the edge. If the edge has the same loop or face on both sides, the two entries in the return list will be the same. If the edge is on the boundary of a sheet body, one of the entries in the returned list will be NULTAG. If the edge is from a wire body, both the entries will be NULTAG.

When the fins of an edge are requested, the left fin of the edge is returned first. The left fin is always in the direction of the edge. If the edge is on the boundary of a sheet body, one of the entries in the returned list will be NULTAG.

If the vertices of an edge or fin are requested, and there are two, the start vertex is returned first and followed by the end vertex.

The edges of a vertex are returned in anti-clockwise order looking at the vertex from outside the body (or inside the void if the vertex is in an interior shell). Any edge which starts and ends at the given vertex will occur twice in the returned list. If the vertex is on the boundary of a sheet body, the first and last edges in the list will themselves be boundary edges of the sheet.

The fins of a vertex are also returned in anti-clockwise order looking at the vertex from outside the body (or inside the void if the vertex is in an interior shell). Only those fins for which the vertex is the end vertex will appear in the list.

Similarly the edges and fins of a loop are returned in anticlockwise order. Edges with the loop on both sides (e.g. wire edges) will occur twice in the list.

The list of vertices of a loop is the list of end vertices of the edges of the loop (in the same order). The only exception to this rule is a loop composed of a single ring edge; such a loop will return a single edge and no vertices.

When the shells of a body are requested the first shell returned is always the exterior shell of the body.

In other cases the order of the returned items is not significant and there will be no duplicated items or NULTAGs in the list.

If 'contyp' is TYGESU, TYGECU or TYGEPT and 'entity' is a body or assembly IDCOEN returns only the appropriate construction geometry attached to 'entity'.

TYOWNR may be used in two kinds of case:

- TYTOIN from an assembly will give the instances IN the assembly; use TYOWNR to get instances OF the assembly. Similarly TYTOAS from an instance will give the assembly (if any) referenced by the instance; use TYOWNR to get the owning assembly.

  This problem does not arise with bodies; TYTOIN and TYTOAS both go upwards to the referencing instances and their owning assemblies.

- Construction geometry, attributes, features and lists may be attached to various types of entity. Use TYOWNR to find the owner irrespective of its type.

If IDCOEN completes without error, it will always return a list in 'entys', even if the list has no entries.

It is now possible to represent edges with either a single curve or two fin curves. For the fin curve case, no curve is associated directly with the edge and so no curve is returned. The relevant curves can be accessed via the fins.

If the given type combination matches a blank cell in the table above, KI_bad_type_combn will be returned in 'ifail'.

Can be called from the GO.

IDCOFE - Identify curve of edge

Receives

```
KI_tag_edge            *edge       edge
```

Returns

```
KI_tag_curve           *curve      curve
KI_cod_error           *ifail      failure indicator
```

Specific Errors

```
KI_missing_geom    The given edge has no associated curve
```

Description The curve associated with the given edge is returned.

It is now possible to represent edges with either a single curve or fin curves. This function only supports the single curve case because this is the only case for which the curve is associated with edge. OUGEEF should be used for accessing the fin curves.

Can be called from the GO.

IDENID - Identify entity by identifier

Receives

```
KI_tag_part   *contxt  entity giving context for search
KI_int_id     *id      id of required entity
KI_cod_ty     *srtype  type of required entity ( one of TYTOIN,
                       TYTORG, TYTOSH, TYTOFA, TYTOLO, TYTOED,
                       TYTOVX, TYGESU, TYGECU, TYGEPT, TYGETF,
                       TYADAT, TYADLI, TYADFE )
```

Returns

```
KI_tag_entity      *entity                 required entity
KI_cod_error       *ifail                  error code
```

Specific Errors

```
KI_bad_type_combn     'srtype' inconsistent with 'contxt'
KI_bad_type           'srtype' is invalid
KI_not_found          'id' not found
```

Description The part 'contxt' is searched for an entity with identifier 'id' of type 'srtype'; this entity is returned as 'entity'. The acceptable combinations of entity and context are as follows :

Entity	Context
Instance. transform	Assembly
Shell, region, face, loop, edge, vertex	Body
Surface, curve, point attached to geome try	Body

Surface, curve, point (construction)	Body or assembly
Feature, attribute, list	Body or assembly

Since IDENID incurs a significant overhead, it should not be used for operations where efficiency is of critical importance; such operations should make direct use of tags to access model entities.

See OUIDEN for further explanation about identifiers.

Can be called from the GO.

IDFSEN - Identifies facesets of one or two bodies

Receives

```
KI_tag_body              *target       target body
<KI_tag_body>            *tool         tool body
<KI_tag_list_edge>       *targed       list of common edges on target
<KI_tag_list_edge>       *tooled       list of common edges on tool
<KI_tag_list_vertex>     *targvx       list of common points on target
<KI_tag_list_vertex>     *toolvx       list of common points on tool
<KI_int_nitems>          *nopts        number of options
KI_cod_idop               opts[nopts]  option codes
<KI_tag_list_topology>   *topol        topology in selected facesets
```

Returns

```
KI_tag_list_list         *targsu       surviving facesets on target
KI_tag_list_list         *toolsu       surviving facesets on tool
KI_tag_list_edge         *targbo       bounds of facesets on target
KI_tag_list_edge         *toolbo       bounds of facesets on tool
KI_tag_list_list         *targrj       rejected facesets on target
KI_tag_list_list         *toolrj       rejected facesets on tool
KI_cod_error             *ifail        failure code
```

Specific Errors

```
KI_inconsistent_facesets  failure to identify facesets
KI_non_manifold           boolean would result in non manifold body
KI_unsuitable_topology    an item in topol is from boundary or wrong
                          body
KI_topol_not_from_body    edge/vertex is not from target/tool as
                          expected
KI_wire_body              target or tool is wire body
KI_contradictory_request  lists of common vertices are not the same
                          length; lists of common edges are not the
                          same length; no boolean option supplied with
                          two bodies; boolean option supplied with no
                          tool; no topology given for selected faceset;
                          more than one boolean option supplied
KI_general_body           general body
```

Description If two bodies are given, with loops of coincident edges and points of contact, the facesets that would survive a boolean (unite, intersect or subtract) are determined. To further restrict the parts of the tool that survive, facesets can be filtered out by giving one or more pieces of topology from each faceset. Each piece of topology can be a face, edge or vertex in the tool and should be in a faceset that would survive the selected boolean; if it is an edge or vertex it should not be on the boundary of the faceset. If any facesets are given, all facesets in the tool that are not selected facesets will not survive and the target facesets will be adjusted accordingly. Coincident edges (and points) must be given in the same order in both lists, it is the users responsibility to check this. If the result of the boolean, taking into account any selected facesets, would be non-manifold then the ifail KI_non_manifold will be returned.

If no tool is given, then the target will be divided into facesets by the edges given, all facesets will be returned as surviving unless facesets are selected. Facesets are selected as above, except the topology must be from the target body. When no tool is given it is impossible to simulate a boolean.

Any void inside the target (or tool) will be a separate faceset (or facesets) and its survival will be determined as for other facesets. If facesets are selected, then a void in the tool (or target if only one body) will not survive unless it is selected with a piece of topology.

Any rubber faces will be ignored and will not be in any returned faceset. Facesets are not 'connected' across rubber faces.

The surviving and rejected facesets of the target (and tool if appropriate) are returned as lists of lists. Each list is the list of faces in one faceset. If there is no tool, the appropriate lists of lists will be set to null.

The returned edges will be the list of edges that divide the surviving and rejected facesets of the target, and also a matching list of edges on the tool (or null if there is no tool). These edges will be some or all of the common edges received.

The options for IDOP are:

Token	Meaning
IDOPUN	Simulate unite. There must be a tool body given.
IDOPIN	Simulate intersection. There must be a tool body given.
IDOPSU	Simulate subtraction. There must be a tool body given.
IDOPFS	Select one or more facesets to survive. One or more pieces of topology must be given in 'topol' to specify the facesets. If a tool is given, the topology must be in the tool.

This function is not supported for general bodies.

IDKYPA - Identify keyed parts

Receives

```
KI_int_nchars       *keylen         length of key
KI_chr_key           key[keylen]    key of part to find
```

Returns

```
KI_tag_list_part    *parts          parts found with given key
KI_tag_list_int     *states         states of parts found
KI_int_nitems       *nparts         number of parts found
KI_cod_error        *ifail          failure indicator
```

Specific Errors

```
KI_key_not_found      part with key not found in memory
```

Description IDKYPA returns a list of the parts with the given key which are attached to the world. A list of the states of the parts is also returned.

All parts in the world are scanned until a stored, modified or unloaded part with the given key is found. New and anonymous parts do not have keys and cannot be found with this function.

There may actually be more than one part with the same key. They are either all modified or one is stored/unloaded. In this situation the tags of all the parts will be returned.

If no part with the given key is attached to the world, KI_key_not_found will be returned in 'ifail'.

IDLSID - Identify entities by identifier

Receives

```
KI_tag_part         *contxt entity giving context for search
KI_tag_list_int     *ids    ids of required entities
KI_cod_ty           *srtype type of required entities ( one of TYTOIN,
                            TYTORG, TYTOSH, TYTOFA, TYTOLO, TYTOED,
                            TYTOVX, TYGESU, TYGECU, TYGEPT, TYGETF,
                            TYADAT, TYADLI, TYADFE )
```

Returns

```
KI_tag_list_entity *entys  required entities
KI_int_nitems      *nitems number of entries in 'entys'
KI_cod_error       *ifail  error code
```

Specific Errors

```
KI_bad_type_combn     'srtype' inconsistent with 'contxt'
KI_bad_type           'srtype' is invalid
```

Description The part 'contxt' is searched for entities with identifiers in the list 'ids' and of type 'srtype'. These entities are returned in 'entys'. A NULTAG will be returned in the list if an identifier is not found.

See IDENID for acceptable combinations of context and entity type.

See OUIDEN for further explanation about identifiers.

Can be called from the GO.

IDNCEN - Identify number of connected entities

Receives

```
KI_tag_entity        *entity    entity to look in
KI_cod_ty            *contyp    type of desired entities
```

Returns

```
<KI_int_nitems>      *nentys    number of entities found
KI_cod_error         *ifail     failure indicator
```

Specific Errors

```
KI_bad_type_combn             bad type combination
KI_general_body               general body
```

Description IDNCEN is identical to IDCOEN except it only returns the number of connected entities.

The valid combinations of the type 'entity' and 'contyp' can be found by examining the table given for the function IDCOEN. In cases where IDCOEN may return duplicated items, these will be included in the count; but where IDCOEN may return NULTAG this is not counted.

This function is not supported for general bodies.

If the given type combination matches a blank cell in the table given with IDCOEN, KI_bad_type_combn will be returned in 'ifail'.

Can be called from the GO.

IDPOFV - Identify point of vertex

Receives

```
KI_tag_vertex          *vertex      vertex
```

Returns

```
KI_tag_point           *point       point
KI_cod_error           *ifail       failure indicator
```

Specific Errors

```
KI_missing_geom   The given vertex has no associated point
```

Description The point associated with the given vertex is returned.

Can be called from the GO.

IDSCEN - Identify single connected entity

Receives

```
KI_tag_entity         *entity    entity to look in
KI_cod_ty             *contyp    type of desired entity
```

Returns

```
<KI_tag_entity>       *conent    entity found
KI_cod_error          *ifail     failure indicator
```

Specific Errors

```
KI_bad_type_combn              bad type combination
KI_general_body                general body
```

Description IDSCEN is similar to IDCOEN except it returns one entity or NULTAG in 'conent' rather than a list of entities.

The valid combinations of the type of 'entity' and 'contyp' can be found by examining the table given for the function IDCOEN; only those combinations marked by 1 or 1* are valid for IDSCEN.

This function is not supported for general bodies.

If, in the type combination table given with IDCOEN, the cell for the type combination given is marked other than 1 or 1*, KI_bad_type_combn will be returned in 'ifail'.

Can be called from the GO.

IDSCLS - Identify single connected entities of a list of entities

Receives

```
KI_tag_list_entity      *entys    entities to look in
KI_cod_ty               *contyp   type of desired entity
```

Returns

```
KI_tag_list_entity      *conent   entities found
KI_int_nitems           *nitems   number of items returned
KI_cod_error            *ifail    failure indicator
```

Specific Errors

```
KI_bad_type_combn             bad type combination
KI_general_body               general body
```

Description IDSCLS is similar to IDCOEN except it returns one entity or NULTAG in 'conent' for every entity in the list 'entys'. These entities may be from different parts.

The valid combinations of the type of entities in 'entys' and 'contyp' can be found by examining the table given for the function IDCOEN; only those combinations marked by 1 or 1* are valid for IDSCLS.

This function is not supported for general bodies.

If, in the type combination table given with IDCOEN, the cell for the type combination given is marked other than 1 or 1*, KI_bad_type_combn will be returned in 'ifail'.

Can be called from the GO.

IDSOFF - Identify surface of face

Receives

```
KI_tag_face                        *face            face
```

Returns

```
KI_tag_surface  *surfac  surface
KI_cod_logical  *revers  true if face normal opposed to surface normal
KI_cod_error    *ifail   failure indicator
```

Specific Errors

```
KI_missing_geom   The given face has no associated surface
```

Description The surface associated with the given face is returned.

'revers' is set to true if the face normal is anti-parallel to the surface normal. If the direction of the face normal is parallel to the surface normal 'revers' is set to false.

Can be called from the GO.

IMPRNT - Imprint bodies or lists of faces

Receives

```
KI_tag_list_entity   *target      target body or list of faces
KI_tag_list_entity   *tool        tool body or list of faces
<KI_int_nitems>      *nopts       number of imprinting options
KI_cod_imop           opts[nopts] option codes
```

Returns

`<KI_tag_list_edge>`	`*targed`	corresponding edges on target
`<KI_tag_list_edge>`	`*tooled`	corresponding edges on tool
`<KI_int_nitems>`	`*nedges`	number of edges
`<KI_tag_list_vertex>`	`*targvx`	corresponding vertices on target
`<KI_tag_list_vertex>`	`*toolvx`	corresponding vertices on tool
`<KI_int_nitems>`	`*nverts`	number of vertices
`KI_cod_error`	`*ifail`	failure code

Specific Errors

`KI_cant_do_imprint`	imprint failure
`KI_tool_faces_many_bodies`	tool faces are from more than one body
`KI_targ_faces_many_bodies`	target faces are from more than one body
`KI_wire_body`	target or tool is wire body or face of wire body
`KI_missing_geom`	target or tool has incomplete geometry
`KI_same_tool_and_target`	target and tool are from the same body

Description Takes a target entity and tool entity and adds edges and vertices where the faces intersect (except where they already exist). The corresponding imprinted (or original) edges and vertices are returned in matching lists.Vertices are returned only where they are isolated points of contact between a face from the target and a face from the tool.

The target and tool each can be a sheet or solid body or a list of faces from a sheet or solid body.

The options for IMPRNT are:

IMOPNT: Do not imprint on tool. Edges and vertices are imprinted on the target only and the lists of edges and vertices on the tool will be returned as null.

IMOPOA: Imprint boundaries of overlapping areas. This is necessary only when the target and tool are faces or sheet bodies. If a face from the target and a face from the tool have identical surfaces, the boundaries of the overlapping area will be imprinted. If this option is not selected faces with identical surfaces will not imprint on each other.

IMOPEF: Extend face list on target. If the imprinting results in incomplete loops of imprinted edges, additional faces in the target will be intersected with the tool and imprinted in an attempt to form complete loops. This will have no effect when the whole target body is supplied. No additional faces in the tool will be used.

This function does not support general bodies and faces belonging to general bodies.

INCUCU - Intersect two curves

Receives

`KI_tag_curve`	`*cu1`	curve 1
`KI_vec_position`	`bound1[2]`	start and end of curve 1
`KI_tag_curve`	`*cu2`	curve 2
`KI_vec_position`	`bound2[2]`	start and end of curve 2
`<KI_tag_surface>`	`*surf`	surface containing both curves
`KI_dbl_box`	`intbox[6]`	box of interest

Returns

```
<KI_tag_list_dbl> *intpts    points of intersection
<KI_tag_list_dbl> *ipars1    parameters of intersections on 'cu1'
<KI_tag_list_dbl> *ipars2    parameters of intersections on 'cu2'
<KI_tag_list_int> *incods    tokens describing intersections
<KI_int_nitems>   *nintpt    number of points returned
KI_cod_error      *ifail     failure indicator
```

Specific Errors

```
KI_cant_do_intersect cant perform requested intersection
KI_bad_end_points    curve not defined between start and end points
KI_not_on_curve      start or end point not on curve
KI_not_on_surface    curve is not on the given surface
KI_invalid_geometry  surface fails checks
```

Description INCUCU finds the intersections between specified regions of two curves. It returns a list of intersection coordinates, two lists containing the parameters of the curves at the intersections, and a list of tokens classifying the intersections. The intersections are ordered along the first curve, 'cu1', and are classified according to the direction of this curve.

The intersection coordinates are returned in a list of length 3*nintpt; if there are no intersections of the curves within the bounds, 'nintpt' is returned as zero and 'intpts' is returned as the NULTAG.

The parameters of 'cu1' at the intersections are returned as a list of length 'nintpt', as are the parameters of 'cu2'.

The intersection types are returned in a list of length 'nintpt'. There are three types of intersection, given by the tokens CICLSI, CICLSC, and CICLEC.

Token	Meaning
CICLSI	A simple intersection not adjoining a region of coincidence. This includes tan gent intersections.
CICLSC	An intersection at the start of a region of coincidence.
CICLEC	An intersection at the end of a region of coincidence.

Whether an intersection is at the start or end of a region of coincidence (CICLSC or CICLEC) is determined by the direction of the first curve, 'cu1', passed to INCUCU.

Coincident intersections (CICLSC and CICLEC) are returned for all points lying at the bounds of regions of coincidence, including the ends of fully coincident curves (or regions of curves). In the case where two closed curves are coincident, but the bounds of 'cu1' are not coincident with those of 'cu2', the intersections returned will be at the bounds of 'cu1'.

The regions of the curves which are to be intersected are specified by giving start and end coordinates on each. These are given in the arrays 'bound1' and 'bound2', in the order start and then end. If these points are the same, and the geometry of the curve is closed, the complete curve will be used. For a trimmed curve the supplied bounds are ignored. However valid vectors should be supplied for the curve in question.

A surface may be supplied which contains both the curves, otherwise it should be NULTAG. The function will work without this surface but it may be more efficient to supply it.

'intbox' describes the box that contains the area of interest. All intersections inside this box will be returned, but ones outside it may not be. The box may be used for efficiency. It is specified as follows:

Its 1st element contains the minimum x_component of the area of interest

Its 2nd element contains the minimum y_component of the area of interest

Its 3rd element contains the minimum z_component of the area of interest

Its 4th element contains the maximum x_component of the area of interest

Its 5th element contains the maximum y_component of the area of interest

Its 6th element contains the maximum z_component of the area of interest

For a box to be valid the difference between the maximum and minimum components in all three principal directions must be greater than or equal to zero.

Any B-curve, B-surface or offset surface must be capable of passing the checks imposed by CHCKEN. The self intersection check is only performed if the appropriate option is set (see SEINTP).

INCUFA - Intersect curve and face

Receives

```
KI_tag_curve           *cu       curve
KI_vec_position         bound[2] start and end of curve
KI_tag_face            *face     face
```

Returns

```
<KI_tag_list_dbl>      *intpts   points of intersection
<KI_tag_list_dbl>      *cuparm   curve parameters at ints.
<KI_tag_list_dbl>      *suparm   surface parameters at ints.
<KI_tag_list_int>      *incods   tokens describing intersections
<KI_tag_list_<entity>> *topol    entity intersected
<KI_int_nitems>        *nintp    number of points returned
KI_cod_error           *ifail    failure indicator
```

Specific Errors

```
KI_cant_do_intersect  cant perform requested intersection
KI_invalid_geometry   curve fails checks
KI_not_on_curve       start or end point not on curve
KI_bad_end_points     curve not defined between start and end points
KI_missing_geom       face lacks geometry
```

Description INCUFA finds the intersections between a bounded curve and a face. It returns a list of intersection coordinates and a list of tokens classifying the intersections. If the bounded curve passes through an edge or vertex of the face the tag of the appropriate entity is also returned. If the bounded curve does not intersect the face, no tag information is returned, but is deemed to be a successful operation.

The intersections are ordered along the bounded curve, and are classified according to the direction of the curve.

The interval of the curve to be intersected with the face is specified by giving the start and end coordinates. These are given in the array 'bound', in the order start and then end. If these points are the same, and the geometry of the curve is closed, the complete curve will be used and the end points will not be treated specially below - i.e. information as to whether they lie in or on the face will not be returned as this is merely a method of specifying the whole curve. If the curve is a trimmed curve the supplied bounds are ignored. However valid vectors should be supplied.

The intersection coordinates are returned in a list of length 3*'nintpt'; if there are no intersections of the curve and face within the interval of the curve bounded by 'bound', 'nintpt' is returned as zero and 'intpts' is returned as the NULTAG. If 'nintpt' intersections are found the list 'cuparm' contains 'nintpt' corresponding curve parameters and 'suparm' contains the corresponding u,v pairs of surface parameters. The curve and surface parametrisations are defined in the documentation of ENCUPA and ENSUPA.

Any edges or vertices intersected by the curve are returned in a list ('topol') of length 'nintpt'.

The intersection types are returned in a list ('incods') of length 'nintpt'. There are two codes for points of intersection when the curve and the surface of the face intersect in a single point and eleven codes for intersections where all or part of the curve lies in the surface.

Single point intersections between bounded curve and surface of face:

Code	**Possible entry in `topol'**	**Description**
CFCLSI	Edge or vertex or NULTAG	The curve passes through the surface of the face
CFCLTG	Edge or vertex or NULTAG	The curve touches the surface of the face but does not pass through the surface.

Intersections with the bounded curve lying in the surface of the face:

Code	**Possible entry in `topol'**	**Description**
CFCLEF	Edge or vertex	The curve passes from being outside the face to being in its interior.
CFCLLF	Edge or vertex	The curve passrs from being in the interior of the face to being outside the face.
CFCLEB	Vertex	The curve passes from being outside the face to a region of coincidence with its boundary.
CFCLLB	Vertex	The curve passes from a region of coincidence with the boundary of the face to being outside the face.
CFCLEI	Vertex	The curve passes from a region of coincidence with the boundary of the face to being in the inte rior of the face.
CFCLLI	Vertex	The curve passes from being in the interior of the face to a region of coincidence with the boundary of the face.
CFCLTI	Edge or vertex	The curve is tangent to the inside of an edge or passes through a vertex but remains inside the face.

CFCLTO	Edge or vertex	The curve is tangent to the outside of an edge or passes through a vertex but remains outside the face.
CFCLUC	Edge or vertex or NULTAG	The start or end of the curve is inside the face or lies on its boundary.
CFCLSC	Edge or vertex or NULTAG	The curve enters the face at the start of a region of coincidence with the surface.
CFCLEC	Edge or vertex or NULTAG	The curve leaves the face at the end of a region of coincidence with the surface.

Whether the curve enters or leaves the face is determined by the direction of the curve, 'cu'.

Any B-curve must be capable of passing the checks imposed by CHCKEN. The self intersection check is only performed if the appropriate option is set (see SEINTP).

INCUSU - Intersect curve and surface

Receives

```
KI_tag_curve          *cu        curve
KI_vec_position        bound[2]  start and end of curve
KI_tag_surface        *surf      surface
KI_dbl_box             intbox[6] box of interest
```

Returns

```
<KI_tag_list_dbl>     *intpts    points of intersection
<KI_tag_list_dbl>     *cuparm    curve parameters at ints.
<KI_tag_list_dbl>     *suparm    surface parameters at ints.
<KI_tag_list_int>     *incods    tokens describing intersections
<KI_int_nitems>       *nintpt    number of points returned
KI_cod_error          *ifail     failure indicator
```

Specific Errors

```
KI_cant_do_intersect  cant perform requested intersection
KI_invalid_geometry   curve fails checks, surface fails checks
KI_not_on_curve       start or end point not on curve
KI_bad_end_points     curve not defined between start and end points
```

Description INCUSU finds the intersections between a bounded curve and a surface. It returns a list of intersection coordinates and a list of tokens classifying the intersections. The intersections are ordered along the bounded curve, and are classified according to the direction of the curve.

The intersection coordinates are returned in a list of length 3*'nintpt'; if there are no intersections of the curve and surface within the bounds, 'nintpt' is returned as zero and 'intpts' is returned as the NULTAG. If 'nintpt' intersections are found the list 'cuparm' contains 'nintpt' corresponding curve parameters and 'suparm' contains the corresponding u,v pairs of surface parameters. The curve and surface parametrisations are defined in the documentation of ENCUPA and ENSUPA. The surface parameters in 'suparm' are only valid for simple and touch intersections. For coincident intersections the values in 'suparm' are not defined.

The intersection types are returned in a list of length 'nintpt'. There are four types of intersection, given by the tokens CICLSI, CICLTG, CICLSC, and CICLEC.

Token	Meaning
CICLSI	A simple intersection not adjoining a region of coincidence.
CICLTG	The curve touches the surface at a point.
CICLSC	An intersection at the start of a region of coincidence.
CICLEC	An intersection at the end of a region of coincidence.

Whether an intersection is at the start or end of a region of coincidence (CICLSC or CICLEC) is determined by the direction of the curve, 'cu'. In the current version regions of coincidence will only be reported when the curve given is found to be coincident with the surface at all points within the bounds supplied.

Coincident intersections (CICLSC and CICLEC) are returned for all points lying at the bounds of regions of coincidence, including the ends of the bounded curve, which are coincident with the surface.

The interval of the curve to be intersected with the surface is specified by giving start and end coordinates. These are given in the array 'bound', in the order start and then end. If these points are the same, and the geometry of the curve is closed, the complete curve will be used. If the curve is a trimmed curve the supplied bounds are ignored. However valid vectors should be supplied.

'intbox' describes the box that contains the area of interest. It is specified in the same way as the box passed to INCUCU. The box is used to improve performance. No guarantee is made that all the intersections returned will lie in it.

For a box to be valid the difference between the maximum and minimum components in all three principal directions must be greater than zero.

Any B-curve or B-surface or offset surface must be capable of passing the checks imposed by CHCKEN. The self intersection check is only performed if the appropriate option is set (see SEINTP).

INFAFA - Intersect two faces

Receives

```
KI_tag_face          *face1          face 1
KI_tag_face          *face2          face 2
<KI_int_nitems>      *nopts          number of options supplied
KI_cod_inop           opts[nopts]    array of options
KI_tag_list_dbl       optdat[nopts]  options data
```

Returns

```
<KI_int_nitems>      *npts           number of intersection points
<KI_tag_list_dbl>    *pts            list of intersection pts
<KI_int_nitems>      *nintcu         number of intersection curves
<KI_tag_list_curve>  *intcu          list of intersection curves
<KI_tag_list>        *intty          list of types of int curves
KI_cod_error         *ifail          returned - failure indicator
```

Specific Errors

`KI_cant_do_intersect`	`cant do intersect, failure in intersection routine`
`KI_not_on_curve`	`given point not on intersection curve`
`KI_missing_geom`	`face lacks geometry`
`KI_bad_option_data`	`incorrect number of values for a point, incorrect number of values for a box, incompatible items in option data`
`KI_duplicate_item`	`duplicate item in option data`

Description INFAFA finds the intersections between two faces ('face1' and 'face2'), returning intersection points ('pts') and intersection curves ('intcu').

If the faces are from the same body any curves returned will be created as construction geometry in the body.

If the faces are from different bodies and any of the curves returned are of type TYCUIN (intersection curve) then these curves will refer to copies of the surfaces of faces.

No attempt is made to define surface regions of partial coincidence, and if the surfaces of faces are fully coincident no intersection data will be returned.

The returned points ('pts') will be returned at points where the faces make point contact. 'npts' indicates the number of intersection points. The 'pts' are returned as a list of doubles of length 3*'nintpt'.

The curves returned ('intcu') will be trimmed curves, 'nintcu' indicating the number of curves returned.

'intty' classifies curves as being either:

Token	Meaning
SICLSI	Simple intersection curves.
SICLTG	Tangent intersection curves

The available options ('opts') and related options data ('optdat') are as described below:

The option INOPBX enables a 3-space box of interest to be supplied. The box may be used to improve performance. The intersection curves returned are not guaranteed to lie within the bounds of the box. The box is specified by supplying six doubles in 'optdat':

Its 1st element contains the minimum x_component of the area of interest

Its 2nd element contains the minimum y_component of the area of interest

Its 3rd element contains the minimum z_component of the area of interest

Its 4th element contains the maximum x_component of the area of interest

Its 5th element contains the maximum y_component of the area of interest

Its 6th element contains the maximum z_component of the area of interest

The options INOPPF and INOPPS enable the supplying of parameter boxes for the first and second face respectively. Again these boxes are only used to constrain the region of interest for performance purposes. The intersection curves returned are not trimmed to lie within the boundaries of the parameter box/es. Four parameters should be supplied to define the parameter box, U-min, U-max, V-min and V-max. The values of these parameters should follow the same conventions as CRBYGE,.

The option INOPSI enables a point on a branch of the intersection to be specified (by supplying the x, y, and z coordinates in 'optdat'). For this case only the branch of the intersection on which this point lies will be returned.

INSUFA - Intersects a surface with a face

Receives

```
KI_tag_surface      *surf           surface
KI_tag_face         *face           face
<KI_int_nitems>     *nopts          number of options supplied
KI_cod_inop          opts[nopts]    array of options
KI_tag_list_dbl      optdat[nopts]  options data
```

Returns

```
<KI_int_nitems>     *npts           number of intersection points
<KI_tag_list_dbl>   *pts            list of intersection pts
<KI_int_nitems>     *nintcu         number of intersection curves
<KI_tag_list_curve> *intcu          list of intersection curves
<KI_tag_list>       *intty          list of types of int curves
KI_cod_error        *ifail          returned - failure indicator
```

Specific Errors

```
KI_cant_do_intersect cant do intersect, failure in intersection routine
KI_not_on_curve      given point not on intersection curve
KI_missing_geom      face lacks geometry
KI_bad_shared_entity surface and face from different bodies
KI_bad_option_data   incorrect number of values for a point, incorrect
                     number of values for a box, incompatible items in
                     option data
KI_duplicate_item    duplicate item in option data
```

Description INSUFA finds the intersections between a surface ('surf') and a 'face' returning intersection points ('pts') and intersection curves ('intcu').

The surface ('surf') must be an orphan or owned by the same body as 'face'. If 'surf' is not an orphan any curves will be created as construction geometry in the body.

If 'surf' is an orphan and any of the curves returned are of type TYCUIN (intersection curve) then these curves will refer to 'surf' and a copy of the surface of the face.

No attempt is made to define surface regions of partial coincidence, and if the surfaces are fully coincident no intersection data will be returned.

The returned points ('pts') will be returned at points where the surface and face make point contact. 'npts' indicates the number of intersection points. The 'pts' are returned as a list of doubles of length 3*'nintpt'.

The curves returned ('intcu') will be trimmed curves, 'nintcu' indicating the number of curves returned.

'intty' classifies curves as being either:

Token	Meaning

SICLSI	Simple intersection curves.
SICLTG	Tangent intersection curves

The available options ('opts') and related options data ('optdat') are as described below:

The option INOPBX enables a 3-space box of interest to be supplied. The box may be used to improve performance. The intersection curves returned are not guaranteed to lie within the bounds of the box. The box is specified by supplying six doubles in 'optdat':

Its 1st element contains the minimum x_component of the area of interest

Its 2nd element contains the minimum y_component of the area of interest

Its 3rd element contains the minimum z_component of the area of interest

Its 4th element contains the maximum x_component of the area of interest

Its 5th element contains the maximum y_component of the area of interest

Its 6th element contains the maximum z_component of the area of interest

The options INOPPF and INOPPS enable the supplying of parameter boxes for the face and surface respectively. Again these boxes are only used to constrain the region of interest for performance purposes. The intersection curves returned are not trimmed to lie within the boundaries of the parameter box/es. Four parameters should be supplied to define the parameter box, U-min, U-max, V-min and V-max. The values of these parameters should follow the same conventions as CRBYGE.

The option INOPSI enables a point on a branch of the intersection to be specified (by supplying the x, y, and z coordinates in 'optdat'). For this case only the branch of the intersection on which this point lies will be returned.

INSUSU - Intersect two surfaces

Receives

```
KI_tag_surface      *surf1         surface 1
KI_tag_surface      *surf2         surface 2
<KI_int_nitems>     *nopts         number of options supplied
KI_cod_inop          opts[nopts]   array of options
KI_tag_list_dbl      optdat[nopts] options data
```

Returns

```
<KI_int_nitems>     *npts          number of intersection points
<KI_tag_list_dbl>   *pts           list of intersection pts
<KI_int_nitems>     *nintcu        number of intersection curves
<KI_tag_list_curve> *intcu         list of intersection curves
<KI_tag_list>       *intty         list of types of int curves
KI_cod_error        *ifail         returned - failure indicator
```

Specific Errors

```
KI_cant_do_intersect     cant do intersect, failed to intersect
KI_not_on_curve          given point not on intersection curve
```

KI_bad_shared_entity	invalid combination of surface owners
KI_not_in_same_partition	surfaces are in different partitions
KI_bad_option_data	incorrect number of values for a point, incorrect number of values for a box, incompatible items in option data
KI_duplicate_item	duplicate item in option data

Description INSUSU finds the intersections between two surfaces ('surf1' and 'surf2'), returning intersection points ('pts') and curves of intersection ('intcu').

The two surfaces must both be orphans or from the same body. In the second case any resulting curves will be created as construction geometry on the body.

No attempt is made to define surface regions of partial coincidence, and if the surfaces are fully coincident no intersection data will be returned.

The returned points ('pts') will be returned at points where the surfaces make point contact. 'npts' indicates the number of intersection points. The 'pts' are returned as a list of doubles of length 3*'nintpt'.

The curves returned ('intcu') will be trimmed curves, 'nintcu' indicating the number of curves returned.

'intty' classifies curves as being either:

Token	**Meaning**
SICLSI	Simple intersection curves.
SICLTG	Tangent intersection curves

The available options ('opts') and related options data ('optdat') are as described below:

The option INOPBX enables a 3-space box of interest to be supplied. The box may be used to improve performance. The intersection curves returned are not guaranteed to lie within the bounds of the box. The box is specified by supplying six doubles in 'optdat':

Its 1st element contains the minimum x_component of the area of interest

Its 2nd element contains the minimum y_component of the area of interest

Its 3rd element contains the minimum z_component of the area of interest

Its 4th element contains the maximum x_component of the area of interest

Its 5th element contains the maximum y_component of the area of interest

Its 6th element contains the maximum z_component of the area of interest

The options INOPPF and INOPPS enable the supplying of parameter boxes for the first and second surface respectively. Again these boxes are only used to constrain the region of interest for performance purposes. The intersection curves returned are not trimmed to lie within the boundaries of the parameter box/es. Four parameters should be supplied to define the parameter box, U-min, U-max, V-min and V-max. The values of these parameters should follow the same conventions as CRBYGE.

The option INOPSI enables a point on a branch of the intersection to be specified (by supplying the x, y, and z coordinates in 'optdat'). For this case only the branch of the intersection on which this point lies will be returned.

INTBYS - Intersect bodies

Receives

```
KI_tag_body      *targby body to be modified
KI_tag_list_body *tolbys body or list of bodies to modify 'targby'
```

Returns

```
KI_tag_assembly  *assemb assembly of resulting bodies
<KI_int_nitems>  *nbodys number of bodies in 'assemb'
KI_cod_error     *ifail  failure code
```

Specific Errors

```
KI_invalid_bodies          Boolean failure or invalid bodies
KI_partial_coi_found       Boolean failure due to partial coincidence
KI_wire_body               Target or tool is a wire body
KI_missing_geom            Target or tool has incomplete geometry
KI_non_manifold            Non-manifold result
KI_same_tool_and_target    Tool body is also target body
KI_mixed_sheets_solids     Mixture of sheet and solid tool bodies
KI_cant_intsc_solid_sheet  Cant intersect solid target with sheet
                           tool bodies
KI_instanced_tools         Instanced tool bodies
KI_duplicate_tools         Duplication in list of tool bodies
KI_not_in_same_body        Target and tools are not all in the same
                           partition
KI_general_body            General body
```

Description The target body (workpiece) is reduced to those regions of space where it overlaps one of the tool bodies (modifiers). The resulting body or bodies replace the workpiece in the world and are instanced in the new assembly returned.

The modifiers are deleted and the tags in the list 'tolbys' become dead.

If 'targby' was instanced in any assemblies, the instances are now of 'assemb'.

Boolean operations such as intersection cannot be performed unless the bodies are completely specified geometrically.

This function does not support general bodies.

KABORT - Aborts a previously interrupted Kernel operation

Receives

```
KI_cod_slab      *reason   specifies reason for abort :
                             SLABUI for user-interrupt
                             SLABRE for run-time error
                             SLABFE for Frustrum error
```

Returns

```
KI_cod_error     *ifail    error code
```

Specific Errors

```
KI_cant_be_aborted        operation cannot be aborted
```

Description KABORT should only be called from the application's interrupt-handler when it has trapped an interrupt. KABORT is used to abort the interrupted Kernel operation so-as to avoid irrecoverably corrupting the Kernel data structures, and must be called whenever an interrupt is trapped (except where the interrupted operation is allowed to resume to completion, which of course is not legitimate for interrupts caused by run-time errors).

After aborting a Kernel operation by use of KABORT, the application should call ROLBLM to rollback the Kernel to the most recent roll-mark; otherwise the Kernel will be left in an uncertain (and possibly inconsistent) state.

KABORT is the only KI routine which may be called from an interrupt handler whilst a Kernel operation is in an interrupted-state; any other routine called under these circumstances will return error KI_recursive_call.

If KABORT is called when the Kernel has not been interrupted (for example, if the interrupt happened within the application's code), it will simply return (having done nothing) with error KI_not_interrupted.

Assuming that the Kernel has indeed been interrupted, what happens next depends on the nature of the interrupt, as specified by the value of 'reason' and whether the Kernel was executing an unprotected PK function call or a protected PK or KI function call:-

Value of `reason'	**Nature of interrupt**	**What happens**
SLABUI	User-interrupt	KABORT sets an abort flag in the Kernel, then returns to the inter rupt handler, which must then allow the original Kernel operation to resume execution. When the Kernel reaches a consistent state, it will abort the operation and call the application's FABORT routine (see the Parasolid Downward Interfaces manual) if a KI function was executing, or registered error handler if a PK function was executing. FABORT or the error handler can do a longjump to a "safe-point" within the application, alternatively, it may simply return to the Kernel, which will return through the original Kernel call with the error-code KI_aborted (for a KI function) or PK_ERROR_aborted (for a PK function).

SLABRE	Run-time error	This indicates that some run-time error has occurred within the Kernel. In this case, KABORT will not return to the interrupt-handler; for a KI or protected PK function call it will do a longjump back to the original Kernel routine, which will return to the calling routine, normally giving error code KI_run_time_error or PK_ERROR_run_time_error respectively. Very occasionally, error code KI_fatal_error/PK_ERROR_fatal_error may be returned; this indicates that the Kernel has been left in a bad state, and the application should abort the program run without making further calls to the Kernel. FABORT is not called for a run-time error inside a KI function. The user supplied PK error handler will be called for a run-time error in a protected PK function. For an unprotected PK function, a run-time error will lead to the user-supplied error handler being called with the error code PK_ERROR_unhandleable_condition and the error handler MUST longjump over the kernel to a safe-point in the applica tions code. If there is no error handler registered, or if it returns, Parasolid will exit the program.
SLABFE	Frustrum error	This behaves like SLABRE, but the error has occured in the Frus trum, an error code of KI_fru_error is returned.

It is not possible to abort operations invoked by calls to STAMOD, STOMOD, ROLBLM, ROLBFN or ROLSMK, or the corresponding PK functions PK_SESSION_start, PK_SESSION_stop, or any PK_MARK_*, PK_PMARK_* or PK_DELTA_* function. If KABORT is called for a user interrupt within the scope of one of these operations, it will return (having done nothing) with error KI_cant_be_aborted; the interrupt handler must then allow the Kernel to continue the operation to completion. If KABORT is called for a run-time error during one of these operations, it will cause the original KI routine to return with error code KI_fatal_error.

Whilst a call to KABORT for a run-time error will force an immediate exit from the KI, a call for a user-interrupt involves a short delay before the operation is aborted. In the unlikely event that a program fault causes Parasolid to go into an infinite loop, it is possible that a call to KABORT requesting a user-interrupt will be ignored. It is therefore suggested that the application's interrupt-handler should incorporate a timer such that any subsequent interrupt more than (say) five cpu-seconds after the original interrupt is treated as if it were a run-time error (by calling KABORT with token SLABRE) in order to force exit from the KI. The timer should be reset by FABORT or the user-supplied PK error handler (which is called after a successfully aborted KI call) and by the interrupt-handler when it decides to treat an interrupt as a run-time error.

KNITEN - "Knits together" bodies by fusing coincident edges

Receives

```
KI_cod_byty        *type   type of result body (solid/sheet)
```

```
KI_tag_body          *target desired result body
KI_tag_list_edge     *eds1   ) edges to be knitted which survive and
KI_tag_list_edge     *eds2   ) those which are destroyed
KI_cod_logical       *shchk  KI_true if a shell connectivity check
                             required on result body
```

Returns

```
KI_cod_rtkn          *state  ok (RTKNOK) / incomplete (RTKNIN)
<KI_tag_list_edge>   *fldeds list of unknitted edges
<KI_int_nitems>      *nfld   number in above list
KI_cod_error         *ifail  failure code
```

Specific Errors

```
KI_corrupt_body           corrupted target body
KI_bad_type               bad type
KI_wire_body              edges in pattern from a wire body
KI_no_eds_from_target     no edges from target body
KI_instanced_tools        an edge is from an instanced (non-target) body
KI_duplicate_list_item    same edge appears in both lists, duplicate
                          edge in list
KI_not_same_length        mismatch in list lengths of knitting pattern
KI_not_in_same_partition  edges are not all in the same partition
KI_general_body           general body
```

Description This function joins a number of faces or bodies together by fusing coincident edge topology. This process is henceforth referred to simply as "knitting"; it should be noted that it is a purely topological operation which will NOT perform any geometric checks whatsoever.

'target' is the body the application wishes to hold the final result of the knitting operation. All the other body topology to be joined onto 'target' is supplied indirectly in the two lists 'eds1' and 'eds2'.'target' must be a solid or sheet body, and all edges in 'eds1' and 'eds2' must be edges of solids and/or sheets. All bodies except 'target' involved in the knit will be destroyed. The edges in 'eds1' will survive the knit while those in 'eds2' will be destroyed. The outer shell of 'target' will also be the outer shell of the result.

The two edge lists, collectively referred to as a "knitting pattern", specify matching edge pairs to be fused together. These lists must be the same length, and must not contain duplicate entries. In addition, at least one edge from 'target' must appear in the pattern and any edges not in 'target' must not come from an instanced body.

Pairs of edges in corresponding positions in 'eds1' and 'eds2' will be knitted together, provided that these corresponding edges satisfy the following:

- they match up 1:1, i.e. every edge in 'eds1' has ONE partner in 'eds2'
- they have the same number of distinct vertices.
- an edge in sheet body has only one adjacent face and a face in a solid body has one adjacent face with a surface attached and one without.

The 1:1 matching is done automatically by other routines in the most common uses of KNITEN. In Partial Booleans the pattern edge lists are made up by IMPRNT and then further filtered by IDFSEN. CRKNPA can also be used to infer the connectivity of an assemblage of sheets. The application can, of course, create a knitting pattern itself, though it then runs the risk of creating an inconsistent body if geometric tolerance requirements were not met.

The return list 'fldeds' will contain all those edges of 'eds1' which did not knit together because of a failure to meet the conditions described above. In the case where a solid body was required ('type' BYTYSO) 'fldeds' will in addition contain all edges in the result body which have only one adjacent face with a surface attached. The presence of such edges in a knitted solid is a consequence of the input knitting pattern being incomplete (i.e. edges were omitted from the knitting pattern which were necessary in order to successfully enclose a volume).

The return code 'state' will be set in RTKNIN (knit incomplete) in such cases, though all edges not in 'fldeds' will still have been knitted successfully. 'State' will normally be set to RTKNOK on a successful knit. Note that the ifail KI_corrupt_body should never be returned in normal operation, and indicates either an internal failure or a corrupt input body.

Argument 'type' enables the application to force the type (BYTYSO or BYTYSH) of the result body. If the token BYTYSS is supplied then a solid will be created if possible (i.e. if the knitting operation encloses a volume) and a sheet will be created otherwise.

Argument 'shchk' enables the application to control whether KNITEN attempts to sort the resulting faces into their respective shells. This operation will not need to be done when KNITEN is used within a partial Boolean.

This function is not supported for general bodies.

LEVASS - Level assembly

Receives

```
KI_tag_assembly        *assemb     assembly to level
```

Returns

```
KI_tag_assembly        *result     new levelled assembly
KI_cod_error           *ifail      failure indicator
```

Specific Errors

```
KI_anon_sub_part  instance of anonymous sub-part of stored part
```

Description LEVASS creates a new assembly which contains, for each path from 'assemb' to a body contained in it, an instance of that body with a transform equal to the product of the transforms attached to instances along the path.

LEVASS will always create a new assembly, even if 'assemb' is already level, or empty. 'assemb' is unchanged.

If 'assemb' is stored (ENSTST) and has a body as an anonymous true-sub-part the assembly created by LEVASS would cause 'assemb' to become unisolated. In this case KI_anon_sub_part is returned in 'ifail'.

MASSPR - Mass and related property calculation

Receives

```
KI_tag_list_entity *entity         entity or list of entities
<KI_int_nitems>    *nopts          number of options in 'iopts'
KI_cod_maop         iopts[ nopts ] mass property request options
double             *accrcy         accuracy parameter, 0.0 - 1.0
```

Returns

```
KI_tag_list_dbl    *periph         size of periphery of entity
KI_tag_list_dbl    *amount         size of entity
KI_tag_list_dbl    *mass           mass
KI_tag_list_dbl    *cofg           centre of gravity
KI_tag_list_dbl    *inert          inertia tensor at cofg
KI_cod_error       *fail           failure indicator
```

Specific Errors

```
KI_different_types        Bodies must all be either wire, sheet or solid.
                          Entities in list not all of the same type
KI_mass_eq_0              Bodies have zero total mass
KI_density_le_0           A body has zero or negative density
KI_missing_geom           A topological entity lacks geometry
KI_request_not_supported Unimplemented mass property requested
KI_empty_assy             No bodies referenced by list of assemblies
KI_general_body           General body
KI_wrong_entity_in_list   Entity of wrong type in list
KI_contradictory_request Contradictory or unknown request made
KI_bad_accuracy           Accuracy out of range
```

Description This routine calculates the following geometric properties of certain topological entities:

Type	"Periphery"	"Amount"	"Mass"	"Cofug, inertia"
Edge	-	Length	-	Cofg, Inertia
Face	Circumference	Area	-	Cofg, Inertia
Wire body	-	Length	Mass	Cofg, Inertia
Sheet body	Circumference	Surface area	Mass	Cofg, Inertia
Solid body	Surface area	Volume	Mass	Cofg, Inertia
Face/sheet considered as Solid	Surface area	Volume	-	Cofg, Inertia
Assembly	Total periphery	Total amount	Mass	As for components

No other types are acceptable. If a list of entities is given they must all be of the same type, and the results are totals, except for the center of gravity and inertia, which refer to the collection of entities. For an assembly or list of assemblies, the results take account of all body occurrences in the assemblies or sub-assemblies; these occurrences must all be the same body type.

Results given as '-' are returned as NULTAG.

The mass is the product of the amount and the density; any bodies with no density attribute will be regarded as having density 1.0. Areas are always treated as unsigned.

The amount, mass, center of gravity, and inertia of bodies and assemblies will take account of any void shells.

For bodies or assemblies, 'inert' is the inertia tensor at the center of gravity (with respect to the X, Y, Z axis directions).

Information found:

There are four possible requests which control how mass properties are calculated - these are given in a list of integers, with the number in the list supplied as another argument. The requests control the level of mass properties calculated, whether periphery data is given, what style the errors are presented in and whether the received entities are considered separately or as part of a single entity.

The main mass properties form a hierarchy in that the mass is needed to find the center of gravity, and the center of gravity is needed to find the inertia. One request code controls the level to which calculations are done. If no request for this option is given, the default is MAOPIN.

Request code	**Interpretation**
MAOPNA	No data in this hierarchy is to be found
MAOPAM	Finds the amount and mass of the entities only
MAOPCG	Finds the center of gravity as well
MAOPIN	Finds the moment of inertia as well

The periphery data is controlled separately by another request, which needs only two settings. The default is MAOPPE.

Request code	**Interpretation**
MAOPNP	No periphery data calculated
MAOPPE	Calculates the periphery of the entities

Accuracy of calculation:

If the calculation requested only involves simple geometries, an accurate method of integration is used, and the parameter 'accrcy' is ignored. In complex cases 'accrcy' is used to determine the accuracy of the calculation: it must be in the range 0.0 to 1.0, and as it approaches 1.0, the calculation will be more accurate but slower. The dependence of errors on 'accrcy' is not linear, and values below 0.9 will give only coarse approximations. A great deal of calculation may be required to satisfy an 'accrcy' of 1.0, and in general convergence cannot be guaranteed. Supplying an accuracy parameter of 1.0 for parts involving B-surfaces is not recommended.

Error estimates:

Mass properties can be requested to calculate error estimates for the values found. For each returned list, the initial entries give the calculated value, and subsequent entries give the associated errors. There is the option of having no errors output, of having a single quantity to add or subtract from the answer value, or having intervals specifying the ranges of the errors. For the center of gravity, the latter two would correspond to a sphere centered on the answer vector and a box containing it. The default is to have MAOPNE.

Request code	Interpretation
MAOPNE	No errors are given
MAOPEM	Modulus +/- to give error range
MAOPEI	An interval bracketing the value is given

This table gives the number of doubles returned for each mass property, first for the value and then for the error option.

	Value	No error	Error modulus	Error interval
Periphery	1	0	1	2
Amount	1	0	1	2
Mass	1	0	1	2
Center of Gravity	3	0	1	6
Moment of Iner tia	9	0	1	18

The moment of inertia returned using the 'MAOPEM' option consists of 9 doubles representing the inertia and 1 double giving the error modulus. Using the 'MAOPEI' option, the data returned consists of 9 doubles representing the inertia followed by 9 doubles for the lower limits and then 9 doubles giving the upper limits of inertia.

Considering received entities as a single entity:

It is possible to supply a list of faces or a list of sheet bodies and have them considered as forming the boundary of a single solid.

Request code	Interpretation
MAOPCS	Supplied list of face/sheets will be considered as the boundary of a single solid

Under this option the mass will not be returned, as there is currently no way of unambiguously specifying a density for the calculation. The Moment of Inertia will be returned, but will be calculated without reference to any density attributes attached to the supplied entities, effectively assuming a density of one for the solid of which they form the boundary.

No checking is done that the faces or sheets provided do form a valid solid boundary. If they do not, the the returned results are likely to be meaningless.

Entities are faces or edges:

A list of faces without the MAOPCS option or a list of edges will be treated as a sheet or wire body respectively. Again the density is not specified, so the Moment of Inertia will be calculated with an assumed density of one.

This function does not support general bodies or assemblies which contain a general body.

MENDEN - Mend a model

Receives

```
KI_tag_list_topology     *topol   body or list of edges
KI_cod_logical           *repopt  option to replace all geometry
```

Returns

```
<KI_tag_list_edge>        fixeds  list of fixed edges
<KI_tag_list_vertex>     *fixvxs  list of fixed vertices
<KI_tag_list_edge>       *ftyeds  list of faulty edges
<KI_tag_list_vertex>     *ftyvxs  list of faulty vertices
<KI_tag_list_int>        *edtoks  token list for faulty edges
<KI_tag_list_int>        *vxtoks  token list for faulty vertices
<KI_tag_list_geometry>   *oldgeo  list of discarded geometry
<KI_int_nitems>          *nfixed  number of fixed edges
<KI_int_nitems>          *nfixvx  number of fixed vertices
<KI_int_nitems>          *nftyed  number of faulty edges
<KI_int_nitems>          *nftyvx  number of faulty vertices
<KI_int_nitems>          *nold    number of discarded geometries
KI_cod_rtmd              *retcod  final state of body
KI_cod_error             *ifail   failure code
```

Specific Errors

```
KI_general_body           general body
KI_missing_geom           body has missing geometry
KI_duplicate_list_item    edge appears twice in argument list
KI_not_in_same_body       edges are not all from the same body
KI_wrong_entity           wrong entity in list
```

Description MENDEN attempts to mend a model so that it conforms to Parasolid's requirements for accuracy and consistency between geometry and topology. Such a model might arise for example by attaching geometry to topology created with CRTOBY, or by the creation of a knitted body. Either a whole body or a list of edges from a body may be supplied for mending.

The body to be mended (or the body from which a list of edges is to be mended) may contain rubber faces. An edge may only be rubber provided that at least one of its fins has a curve attached. A vertex may only be rubber provided that at least one fin meeting at it has a curve attached. If these conditions are not satisfied ifail KI_missing_geom will be returned.

The mending operation will check that edge and vertex geometry conforms to Parasolid's requirements for accuracy and consistency, and if it does not, will replace the faulty geometry where possible. If the replacement option 'repopt' is set to true, original geometry will be replaced wherever a Parasolid replacement can be calculated, even if the original geometry satisfies the accuracy requirements. Some limited merging may also be performed, in which case the topology of the body will alter and geometric items attached to merged topology will be deleted.

If 'repopt' is set to false, original geometry will only be replaced where necessary, and no merging will take place. Setting the replacement option to true will increase the chances of success, though the resulting body will not be so close to the original.

MENDEN returns lists to indicate which edges and vertices have been successfully fixed, and which have not, together with the reasons for failure.

'fixeds' contains the list of edges that have been mended successfully and will have 'nfixed' elements.

'fixvxs' contains the list of vertices that have been mended successfully and will have 'nfixvx' elements.

'ftyeds' contains the list of edges which failed to mend and will have 'nftyed' elements.

'ftyvxs' contains the list of vertices which failed to mend and this will have 'nftyvx' elements.

'edtoks' is a list of tokens indicating the reason that each edge in list 'ftyeds' failed to mend. Each edge in 'ftyeds' has a corresponding token in 'edtoks'. Each token will be one of the following:

Token	Meaning
MDFARF	Mend failed because one or more adjacent faces lack geometry.
MDFACS	Mend failed because adjacent faces have coincident surfaces preventing calcula tion of replacement curve.
MDFANS	Mend failed because adjacent surfaces do not intersect.
MDFAFM	Mend failed during calculation of replacement geometry.

'vxtoks' is a list of tokens indicating the reason that each vertex in list 'ftyvxs' failed to mend. Each vertex in 'ftyvxs' has a corresponding token in 'vxtoks'. Each token will be one of the following:

Token	Meaning
MDFANI	Mend failed because there is no common intersection between adjacent beome try. (Intersection points do not lie on all incoming edge curves and underlying face surfaces).
MDFAFE	Mend failed because one or more incoming edges is faulty.
MDFAFM	Mend failed during calculation of replacement geometry.

Original geometry that is replaced during the mending operation is returned in list 'oldgeo'. The length of this list is returned in 'nold'.

'retcod' contains a token which indicates the final state of the body. It will be one of:

Token	Meaning
RTMDMS	Mending was successful - the body now conforms to Parasolid accuracy.
RTMDMF	Mending failed - the body is still invalid.

It is important to note that even when mending is successful, it is still possible for the resultant body to be invalid. A call to MENDEN should therefore always be accompanied by a call to CHCKEN.

Mending also has certain limitations. Since only edge and vertex geometry is recalculated, mending may not succeed at tangent edges if the adjacent surfaces do not meet in a curve. Similarly difficulties may arise at vertices with more than three edges. Edges of sheet bodies can also fail to mend due to complications which arise when projecting edge curves onto surfaces.

Note also that mending will only be completely successful in those cases where the final mended body lies entirely within the size box.

General bodies are not supported.

MERGEN - Remove redundant topology from entity

Receives

```
KI_tag_topology          *entity      entity to merge
```

Returns

```
KI_cod_error             *ifail       failure indicator
```

Specific Errors

```
KI_none_mergeable              no mergeable entities
KI_wrong_entity                entity of wrong type
KI_general_body                general body
```

Description Redundant ('mergeable') edges and vertices are removed from the given entity, which may be a body, face, edge or vertex.

A vertex will normally be mergeable if it is attached to only one edge, or if it is attached to only two edges, with the same curve geometry.

An edge will normally be mergeable if it is a wire edge, or if the faces on either side have the same surface geometry.

The allowed entity types are treated as follows:

- Vertex: - The vertex will be removed if it is mergeable.
- Edge: - If the edge is mergeable, it will be removed. If its vertices are then mergeable, they will be removed.
- Face: - Any mergeable wire edges in the face will be removed. Then any mergeable vertices in or on the boundary of the face will be removed.
- Body: - Mergeable edges and vertices are removed from the body until none remain.

If the body contains B-curves or B-surfaces then connected sets of edges on each B-curve and connected sets of faces on each B-surface will be merged into a single edge or face provided that the B-geometry is capable of passing the composite geometry checks (see CHCKEN). However if any of the B-curves or B-surfaces are not capable of passing the composite geometry checks then these curves and surfaces will be split to produce smaller curves and surfaces that do pass the checks. These will then be merged as above.

Note that the composite geometry checks will only be performed if SLIPCO (see SEINTP) is set to 0.

This function is not supported for general bodies.

NEGENT - Negates (reverses) an entity

Receives

```
KI_tag_entity        *entity     entity to be negated
```

Returns

```
KI_cod_error         *ifail      failure code
```

Specific Errors

KI_unsuitable_entity	unsuitable entity
KI_bad_sharing	entity is a dependent of another geometric entity; dependent is owned by more than one geometric entity
KI_is_attached	geometric entity is attached to topology
KI_general_body	general body

Description Negates (reverses) a curve, surface or body.

Operations performed

- Negating a curve reverses the curve direction.
- Negating a surface reverses the surface normal.
- Negating a body reverses the normals of all non-rubber faces.

If the entity is a curve or surface it must not be attached to any topology. Furthermore curve and surface entities must not be dependents of any other geometric entities.

This function is not at present supported for general bodies.

OFFABY - Offsets faces of a solid or sheet body

Receives

```
KI_tag_body            *body   body to be offset
KI_<dbl>               *offset default offset
KI_cod_logical         *check  level of checking required
<KI_tag_list_face>     *fixed  faces not to be offset
<KI_tag_list_face>     *faces  faces offset by other amounts
<KI_tag_list_dbl>      *dists  list of other offset distances
KI_dbl_distance        *tol    maximum applied tolerance
<KI_int_nitems>        *mxflts maximum number of entities in badtag
```

Returns

```
<KI_tag_list_entity> *badtag entities which caused problems
KI_cod_rtof          *state  state of body after offset
KI_cod_error         *ifail  failure code
```

Specific Errors

KI_cant_offset	failed to offset faces of body
KI_bad_tolerance	proposed tolerance is too small
KI_bad_value	non-default offset too small
KI_not_in_same_body	Offset face is not in supplied body; Fixed face is not in supplied body

KI_duplicate_list_item	face is in both fixed and faces
KI_list_wrong_length	list of faces and dists not same length; too many faces in lists fixed and faces; too many faces for non-default offset; too many fixed faces
KI_list_too_short	no dists supplied for faces
KI_unsuitable_entity	body has more than one shell

Description The 'body' is returned with all or some faces offset. A positive offset indicates an offset outwards (i.e. in the direction of the face normal) and a negative offset an offset inwards.

The default offset to be applied is given in the 'offset' argument. Faces which are not to be offset are supplied in the 'fixed' argument. Faces to be offset by distances other than the default are supplied in the 'faces' argument with the specific distances supplied in the corresponding position in the 'dists' argument. Mixtures of positive and negative distances are allowed.

If 'offset' is zero then only those faces in the 'faces' list will be offset. In this case the 'fixed' list is ignored.

Under some circumstances the function may need to replace exact geometry by tolerant geometry. For instance, in the case of a sheet body some edge geometry at the boundary may have to be approximated by SP curves in order to generate a boundary curve on the offset sheet. In all such situations the new geometry will have a tolerance less than or equal to the tolerance supplied through the 'tol' argument.

Three situations can give rise to changes in the topology of the body :

- Dealing with degeneracies on a sheet body comprised of a single face. For example, a vertex may offset to an edge.
- Dealing with geometry which fails to offset. If it is known that the offset surface of a face would be self-intersecting an attempt is made to remove the face. For example, a blend may be removed from an edge. The investigation of self-intersection is not exhaustive, however, and it can occur that instances are not trapped.
- Dealing with configurations which can be repaired. For instance an edge can offset to a point or a face can become absorbed into the body.

The extent to which checking is applied to the body is specified by the 'check' argument. If 'check' is true then face-face checks are done on the body in addition to default checks. For most applications setting 'check' false will give an adequate level of checking.

The error reporting scheme comprises the four arguments 'badtag', 'state', 'mxflts' and 'ifail'. A non-zero 'ifail' is reserved for reporting unsuitable arguments to the function and system errors.

Algorithmic failures where the item causing the failure can be identified result in a zero 'ifail' and more specific information being returned in the 'state' argument and 'badtag'. For example, if the new geometry cannot be found for an edge 'state' RTOFEM will be returned and the tag of the edge whose curve could not be found in 'badtag'. The user may set an upper limit on the number of faults found in 'mxflys'; if 'mxflys' is zero then no limit is applied.

'state' refers to the validity of the item after modification and not to its original validity and after such a failure 'body' may be corrupt and a rollback should be performed. For this reason tags of new topology cannot be returned in 'badtag' and the tags will refer to adjacent faces in the original body supplied. For example, if a new edge cannot be modified a 'state' RTOFEM is returned and 'badtag' will contain the tag of a list of the two faces whose offsets are to be used to find the new curve. 'badtag' is a list of items which failed for the reason indicated in the 'state' argument.

Possible values of 'state' and the contents of 'badtag' are :

Token	Tag in badtag	Meaning
RTOFOK	null	Body is OK
RTOFSO	face	Surface failed to offset or face coould not be deleted
RTOFVM	vertex	Failed to find newgeometry for vertex
	list faces	Failed to find geometry for new vertex
RTOFEM	edge	Failed to find new geometry for edge
	list faces	Failed to find geometry for new edge
RTOFVT	edge	Edge should have disappeared
	list faces	New edge should have disappeared
RTOFNG	null	Offset body was negative
RTOFFA	face	Face failed checks
RTOFSX	list faces	Pair of faces where face-face inconsistency found

Notice that a successful execution of the offsetting operation is indicated by:

ifail returning KI_no_errors

state returning RTOFOK or RTOFNG

This function is not supported for general bodies.

OUATDF - Output an attribute type definition

Receives

```
KI_tag_attrib_def   *type          attribute type
<KI_int_nitems>     *bufsiz        amount of space available for name
```

Returns

```
KI_chr_string        name[bufsiz]  name of type
KI_int_nitems       *namlen        length of type name
KI_tag_list_int     *option        list of option codes
KI_tag_list_<list>  *opdata        corresponding list of data-lists
KI_cod_error        *ifail         error code
```

Description OUATDF returns the data supplied to CRATDF to define the attribute type, or the equivalent data for types not defined by CRATDF.

Type name and bufsiz:

The value to supply in 'bufsiz' is the number of characters that can legitimately be written to the array 'name' without overrunning the space allocated to it. OUATDF will under no circumstances attempt to write more than 'bufsiz' characters to 'name'. The value returned in 'namlen' is the actual length of the name of the attribute type. If this is less than 'bufsiz' then the name of the attribute type definition is returned in 'name'; otherwise only the first 'bufsiz' characters are returned; if the entire name is required, then the returned value of 'namlen' can be used as the value of 'bufsiz' for a second call to OUATDF which is sure to return the entire name.

Even when the function returns with 'ifail' zero the entire name will not have been returned in 'name' if the value of 'namlen' is greater than that of 'bufsiz'.

Option data:

The description of the type's fields, legal owners and behavior under modelling operations is returned in 'option' and 'opdata'. 'option' contains integer option codes, while 'opdata' contains tags of lists of data, each one relating to the option whose code is in the corresponding position in 'option'. The meanings of the option codes and associated data are explained in the documentation of CRATDF.

Can be called from the GO.

OUBBCO - Output bulletin board controls

Returns

```
KI_tag_list_int      *ents     entity types being bulletined
KI_tag_list_list     *events   their bulletinning events
KI_int_nitems        *nents    length of the above lists
KI_tag_list_int      *opts     current control options
<KI_int_nitems>      *nopts    no of control options
KI_cod_error         *ifail    error code
```

Description OUBBCO outputs the state of the bulletin board's controls in the same form as data is supplied to SEBBCO to set the state, except that arrays are replaced by lists. See SEBBCO for more information.

Thus each list in the list 'events' indicates which events are being recorded for entities of the type whose code is given in the corresponding position in the list 'ents'.

The list 'opts' contains option codes which indicate further information about the state of the bulletin board. The presence of a code in 'opts' indicates that the current state of the bulletin board is the state that the board would be in after a successful call to SEBBCO supplying the given code in 'opts'. In particular, precisely one of BBOPOF (indicating that the bulletin board is off), BBOPON (bulletin board will record tags but not any user field data) and BBOPUF (bulletin board will record tags and user field data) will be present in 'opts'.

OUBBEV - Output full bulletin board information

Receives

```
KI_cod_logical    *empty  true if bulletin board is to be emptied after
                          being output
```

Returns

```
<KI_int_nitems>    *nevent no of recorded events
<KI_tag_list_int> *events recorded event tokens
<KI_tag_list_int> *nperev no of entities at each event
<KI_int_nitems>    *nent   total no of entities recorded
```

```
<KI_tag_list_tag> *ents   entities recorded
<KI_tag_list_int> *enttyp entity types
<KI_tag_list_int> *usflds entity user fields
KI_cod_error      *ifail  error code
```

Specific Errors

```
KI_bulletinb_is_off  cannot read bulletin board when it is off
```

Description Outputs bulletin board information on events occurring to entities since the bulletin board was last emptied. Depending on the value of 'empty', optionally empties the bulletin board after reading it.

The bulletin board data is returned as a list of events and associated with each event there are the tags and types of the entities involved, the number of entities involved and optionally the user-field of each entity.

Whether user-field data is returned depends on the setting of the user-field option in SEBBCO. When user-field data is not returned, the list 'usflds' is returned as the NULTAG.

The list 'events' contains 'nevent' tokens from the sequence BBEV00. The list 'nperev' contains 'nevent' integers which specify the number of entities involved in an event. The event and number per event are in corresponding positions in the lists 'events' and 'nperev'.

The list 'ents' contain 'nents' tags. These are the tags of the entities which have been recorded on the bulletin board. The list 'enttyp' contains 'nents' tokens from the sequences TYTO00, TYGE00 and TYAD00. The token returned in the 'enttyp' list is the type of entity in the corresponding position in the 'ents' list.

The position of the entities corresponding to the event in entry number "n" of the 'events' list can be found by summing the first "n-1" entries of the 'nperev' list and adding 1. The value in the "n"th entry of the 'nperev' list is the number of entities involved in the event in entry number "n" of the 'events' list.

The list 'usflds' contains "'nents' * ufd_size" integers where ufd_size is the user-field size as set by STAMOD.The user-field of the entry number "n" in the 'ents' list can be found in entries "(n-1)*ufd_size+1" to "n*ufd_size" inclusive.

The events merge, split, copy and transfer (types BBEVME, BBEVSP, BBEVCO and BBEVTR respectively) all have more than one entity associated with them. These are in the order:

- Merge - first entity is the entity which is dominant and remains other entities are those which were deleted as part of the operation
- Split - first entity is the entity which was originally split other entities are those which were created as part of the operation
- Copy - first entity is the new entity other entity is the entity of which the previous is a copy
- Transfer - first entity is the entity transferred other entity is the entity which was previously its owner

Two records of the bulletin board may be merged into one under certain circumstances. These are that the events are both of the types create (BBEVCR), delete (BBEVDE), change (BBEVCH), transform (BBEVTF), transfer (BBEVTR) or attribute change (BBEVAC) and that the events are consecutive (that is, no other event has happened to the entity prior to the second event and after the first event).

First Event	Second Event					
	Created	Changed	Transformed	Deleted	Transferred	Att Ch
Created	(1)	Created	Created	(2)	Created	Created
Changed	(1)	Created	(3)	Deleted	(3)	(3)
Transformed	(1)	(3)	Transformed	Deleted	(3)	(3)
Deleted	(1)	(1)	(1)	(1)	(1)	(1)
Transferred	(1)	(3)	(3)	Deleted	Transferred	(3)
Att Ch	(1)	(3)	(3)	Deleted	(3)	Att Ch

where:

(1) this case cannot occur.

(2) the entity will be removed from the bulletin board entirely.

(3) both events will be bulletined

All entries made since the bulletin board was last emptied will be checked in this way.

OUBLSS - Output blending surface

Receives

```
KI_tag_surface       *surfac  blending surface
```

Returns

```
KI_cod_tybl          *bltype  blend type
KI_tag_surface       *ssurf1  first supporting surface
KI_tag_surface       *ssurf2  second supporting surface
<KI_tag_surface>     *ssurf3  third supporting surface
KI_tag_list_int      *iipa12  integer parameters for ssurf1,2
<KI_tag_list_int>    *iipa23  integer parameters for ssurf2,3
<KI_tag_list_int>    *iipa31  integer parameters for ssurf3,1
KI_int_nitems        *nipars  length of int parameter lists
KI_tag_list_dbl      *irpa12  double parameters for ssurf1,2
<KI_tag_list_dbl>    *irpa23  double parameters for ssurf2,3
<KI_tag_list_dbl>    *irpa31  double parameters for ssurf3,1
KI_int_nitems        *nrpars  length of real parameter lists
KI_cod_logical       *sense   surface sense. explained below
KI_cod_error         *ifail   failure code
```

Specific Errors

```
KI_wrong_sub_type           surface is not a blend
```

Description The blending surface definition is output with a type-code, 'bltype', that enables it to be interpreted as follows:

TYBL1B   blended edge

The length of the lists 'iipa12', 'iipa23' and 'iipa31' is given by 'nipars'. In the current version 'nipars' will always be 3. The length of the lists 'irpa12', 'irpa23', and 'irpa31' is given by 'nrpars'. In the current version 'nrpars' will always be 5.

As from version 3.0, offset blends (ITYPE = 1) have been withdrawn.

Contents of the lists 'iipaXY'

Position in list	Contents and its significance
1	ITYPE3 => rolling ball blend

Contents of the lists 'irpaXY' ITYPEs 3 - Rolling ball blends

Position in list	Value and its significance	
1	radius X	RadiusX and radius Y always have the same magnitude but may differ in the sign.
2	radiusY	

The surface junction which is blended is indicated by the signs of the radii. A positive value signifies that the blend lies on the same side of the supporting surface as the supporting surface's normal. (The supporting surface's normal direction is affected by the surface's sense).

Summary of the data returned

Return:	ssurfX			iipXX, irpXX		
	1	**2**	**3**	**12**	**23**	**31**
SSTYPE:						
TYBLLIB	S	S	-	L	-	-

S - Tag of surface

- - Null tag

L - List of length 'nipars' or 'nrpars'

The normal to a blending surface whose sense is KI_true is away from the center of curvature.

OUBSCU - Outputs a curve in B-spline form

Receives

```
KI_tag_curve        *curve          curve
KI_dbl_distance     *tol            tolerance for approximation
<KI_int_nitems>     *nopts          number of options in 'iopts'
KI_cod_srop          iopts[nopts]   conversion options
```

Returns

```
KI_tag_list_dbl     *ctrl           control points
KI_int_dimension    *dim            dimension of control points
KI_int_order        *order          order of bspline curve
KI_int_nitems       *nctrl          number of control points
KI_tag_list_dbl     *knots          knot vector
<KI_tag_list_int>   *props          list of curve properties
<KI_int_nitems>     *nprops         number of curve properties
KI_cod_error        *ifail          failure code
```

Specific Errors

```
KI_tolerances_too_tight         tolerance too tight
KI_cant_make_bspline            failed to convert geometry
```

Description The supplied curve is output in B-spline form.

If the curve is of type 'TYCUTR' (trimmed curve) then the resulting B-spline curve will be bounded by the end points of the trimmed curve.

The B-spline will represent the curve exactly if possible, but it may be necessary to approximate it. If so, 'tol' specifies an upper bound on the distance between the B-spline and the original curve, which this function will usually satisfy although this is not guaranteed. During the approximation process, the curve is sub-divided and each segment approximated by a cubic B-spline curve. If the required tolerance cannot be met when any of these segments is less than about one millionth of the parameter range then the function will fail with error KI_tolerance_too_tight indicating that a larger tolerance may allow a successful approximation.

Representations of lines and the rational forms of circles and ellipses are exact. When the representation of a B-curve is changed it will, in general, only be exact when its order is not decreased.

The format of the output curve is selected by options in 'iopts'. Options allowed are:-

Option	**Description**
SROPCU	Force output of B-splines of degree 3
SROPNR	Force output of non-rational B-splines

The effects of these options on the representations produced for different curve types are:-

	none	**SROPCU**	**SROPCU SROPNR**	**SROPNR**
line	linear	cubic	cubic	linear
circle	rational cubic	rational cubic	cubic	cubic
ellipse	rational cubic	rational cubic	cubic	cubic
B-curve	no change	cubic	cubic	rational to cubic
all other curves*	cubic	cubic	cubic	cubic

* If an intersection curve or SP-curve lies on a B-surface and coincides with a constant parameter line on that B-surface, the equivalent B-curve will be extracted from the surface. In this case, the options will apply to the extracted B-curve. Thus exact representations of these curves will be possible in certain cases.

The meaning of the values in the arguments 'ctrl', 'dim', 'order', 'nctrl' and 'knots' are given under OUBSPC.

Properties list 'props': The following properties may be returned:

- PAPRPE - the B-spline is periodic. If this token is not present then the B-spline is not periodic.
- PAPREX - the B-spline is an exact representation of the original curve. If this token is not present, then the B-spline approximates the original, with the maximum error bounded by 'tol'.

Number of properties 'nprops':

- Gives the number of properties in the 'props' list.

OUBSED - Outputs the curve of an edge in B-spline form

Receives

```
KI_tag_edge          *edge            edge of curve
KI_dbl_distance      *tol             tolerance for approximation
<KI_int_nitems>      *nopts           number of options in 'iopts'
KI_cod_srop           iopts[nopts]    conversion options
```

Returns

```
KI_tag_list_dbl      *ctrl            control points
KI_int_dimension     *dim             dimension of control points
KI_int_order         *order           order of bspline curve
KI_int_nitems        *nctrl           number of control points
KI_tag_list_dbl      *knots           knot vector
<KI_tag_list_int>    *props           list of curve properties
<KI_int_nitems>      *nprops          number of curve properties
KI_cod_error         *ifail           failure code
```

Specific Errors

```
KI_tolerances_too_tight     tolerance too tight
KI_cant_make_bspline        failed to convert geometry
KI_missing_geom             insuffecent geometry to define edge
```

Description Outputs the curve of an edge in B-spline form. If the edge is not toleranced the B-spline curve will be bounded by the end points of the edge. If the edge is toleranced the B-spline curve will represent the primary fin curve if present, otherwise it will represent the secondary fin curve.

The B-spline will represent the curve exactly if possible, but it may be necessary to approximate it. If so, 'tol' specifies an upper bound on the distance between the B-spline and the original curve, which this function will usually satisfy although this is not guaranteed.

Representations of lines and the rational forms of circles and ellipses are exact. When the representation of a B-curve is changed it will, in general, only be exact when its order is not decreased.

The format of the output curve is selected by options in 'iopts'. Options allowed are:-

Option	Description
SROPCU	Force output of B-splines of degree 3
SROPNR	Force output of non-rational B-splines

The effects of these options on the representations produced for different curve types are:-

	none	**SROPCU**	**SROPCU SROPNR**	**SROPNR**
line	linear	cubic	cubic	linear

circle	rational cubic	rational cubic	cubic	cubic
ellipse	rational cubic	rational cubic	cubic	cubic
B-curve	no change	cubic	cubic	rational to cubic
all other curves*	cubic	cubic	cubic	cubic

* If an intersection curve or SP-curve lies on a B-surface and coincides with a constant parameter line on that B-surface, the equivalent B-curve will be extracted from the surface. In this case, the options will apply to the extracted B-curve. Thus exact representations of these curves will be possible in certain cases.

The meaning of the values in the arguments 'ctrl', 'dim', 'order', 'nctrl' and 'knots' are given under OUBSPC.

Properties list 'props': The following properties may be returned:

- PAPRPE - the B-spline is periodic. If this token is not present then the B-spline is not periodic.
- PAPREX - the B-spline is an exact representation of the original curve. If this token is not present, then the B-spline approximates the original, with the maximum error bounded by 'tol'.

Number of properties 'nprops':

- Gives the number of properties in the 'props' list.

OUBSFA - Outputs the surface of a face in B-Spline form

Receives

```
KI_tag_face        *face          face
KI_dbl_distance    *tol           tolerance for approximations
<KI_int_nitems>    *nopts         number of options in 'iopts'
KI_cod_srop         iopts[nopts]  conversion options
```

Returns

```
KI_tag_list_dbl    *ctrl          control points
KI_int_dimension   *dim           dimension of control points
KI_int_order       *uorder        order of surface patches in u
KI_int_order       *vorder        order of surface patches in v
KI_int_nitems      *ncol          number of cols of control points
KI_int_nitems      *nrow          number of rows of control points
KI_tag_list_dbl    *uknots        knot vector in the u direction
KI_tag_list_dbl    *vknots        knot vector in the v direction
<KI_tag_list_int>  *props         list of surface properties
<KI_int_nitems>    *nprops        number of surface properties
KI_cod_error       *ifail         failure code
```

Specific Errors

KI_cant_make_bspline	failed to convert geometry
KI_missing_geom	insufficent geometry to define extent of face
KI_tolerances_too_tight	tolerance too tight

Description This function outputs the surface of a face in B-spline form. The surface output will be large enough to contain the face.

The B-spline will represent the surface exactly if possible, but it may be necessary to approximate it. If so, 'tol' specifies an upper bound on the distance between the B-spline and the original surface, which this function will usually satisfy although this is not guaranteed.

Representations of planes and the rational forms of cylinders, cones, spheres and tori are exact. When the representation of a B-surface is changed it will, in general, only be exact when its order is not decreased.

The format of the output surface is selected by options in 'iopts'. Options allowed are:-

Option	Description
SROPCU	Force output of B-splines of degree 3
SROPNR	Force output of non-rational B-splines

The effects of these options on the representations produced for different surface types are:-

Planes

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	2	4	4	2
`vorder'	2	4	4	2
`dim'	3	3	3	3

Cylinders and Cones

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	4	4	4	4
`vorder'	2	4	4	2
`dim'	4	4	3	3

Spheres and Tori

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	4	4	4	4
`vorder'	4	4	4	4
`dim'	4	4	3	3

B-surfaces

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	* or 4	4	4	* or 4
`vorder'	* or 4	4	4	* or 4
`dim'	* or 3	* or 3	3	3

Where * fields are taken from surface definition. The value of "* or n" fields is taken from the surface definition if the representation is exact, otherwise the value n is used.

Swept surfaces

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	*	4	4	*
`vorder'	2	4	4	2
`dim'	*	* or 3	3	3

Where * fields are derived from the curve which was swept. The value of "* or n" fields is derived from the swept curve if the representation is exact, otherwise the value n is used.

Spun surfaces

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	*	4	4	4
`vorder'	4	4	4	4
`dim'	4	4	3	3

Where the * field is derived from the curve which was spun.

Offset surfaces

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	*	4	4	*
`vorder'	*	4	4	*
`dim'	*	*	3	3

Where * fields are the same as for the surface which was offset.

All other surfaces

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	4	4	4	4
`vorder'	4	4	4	4
`dim'	3	3	3	3

The meaning of the values in the arguments 'ctrl', 'dim', 'uorder', 'vorder' 'ncol', 'nrow', 'uknots' and 'vknots' are given under OUBSPS.

Properties list 'props': The following tokens may be returned:

- PAPRPU - the B-spline surface is periodic in u
- PAPRPV - the B-spline surface is periodic in v. The default is that the B-spline surface is not periodic.
- PAPREX - the B-spline surface represents the original surface exactly. If this token is not present then the B-spline approximates the original.

Number of properties 'nprops':

- Gives the number of properties in the 'props' list.

OUBSPC - Output B-curve in B-spline form

Receives

```
KI_tag_b_curve        *bc       B-curve
```

Returns

```
KI_tag_list_dbl       *ctrl     control points
KI_int_dimension      *dim      dimension of control points
KI_int_order          *order    order of curve
KI_int_nitems         *nctrl    number of control points
KI_tag_list_dbl       *knots    knot values
<KI_tag_list_int>     *props    list of curve properties
<KI_int_nitems>       *nprops   number of curve properties
KI_cod_error          *ifail    failure indicator
```

Description This function outputs a B-curve in B-spline form.

Dimension of control points 'dim':

- For rational curves 'dim'=4.
- For non-rational curves 'dim'=3.

Order of the curve 'order':

- The order of the curve = degree + 1.
- The minimum order is 2.

Control points 'ctrl':

- This is a real list containing 'nctrl' vectors of dimension 'dim'.
- For non-rational curves, the vectors are points in 3-space and are returned [x0,y0,z0,x1,y1,z1,...].
- For rational curves each vector contains a point in 3-space followed by a weight for the point. The points are returned [x0,y0,z0,w0,x1,y1,z1,w1,...]. The weights are positive.

Number of control points 'nctrl':

- For non-periodic curves 'nctrl' >= 'order'.
- For periodic curves 'nctrl' >= 3.

Knot vector 'knots':

- The knot values form a non-decreasing sequence.
- For non-periodic curves there are ('nctrl' + 'order') knot values.
- For periodic curves there are ('nctrl' + 1) knot values.
- The knot values follow the rules described in CRBSPC, see page .

Properties list 'props': At most one property token will be returned:

- PAPRPE - the curve is periodic. If this token is not present then the curve is not periodic.

Number of properties 'nprops':

- Gives the number of properties in the 'props' array.
- There is only one property that could apply (periodicity) and so 'nprops' is always either 0 or 1.

OUBSPS - Output B-surface in B-spline form

Receives

```
KI_tag_b_surface    *bs      B-surface
```

Returns

```
KI_tag_list_dbl     *ctrl    control points
KI_int_dimension    *dim     dimension of control points
KI_int_order        *uorder  order of surface in u
KI_int_order        *vorder  order of surface in v
KI_int_nitems       *ncol    number of cols of control points
KI_int_nitems       *nrow    number of rows of control points
KI_tag_list_dbl     *uknots  knot vector in the u direction
KI_tag_list_dbl     *vknots  knot vector in the v direction
<KI_tag_list_int>   *props   list of surface properties
<KI_int_nitems>     *nprops  number of surface properties
KI_cod_logical      *sense   surface sense
KI_cod_error        *ifail   failure indicator
```

Description This function outputs a B-surface in B-spline form.

Dimension of control points 'dim':

- For rational surfaces 'dim'=4.
- For non-rational surfaces 'dim'=3.

Order of the surface in u and v 'uorder' and 'vorder':

- The order = degree + 1.
- The minimum order is 2.

Control points 'ctrl':

- This is a real list containing 'ncol'*'nrow' vectors of dimension 'dim'.
- The vectors are output row by row.
- For non-rational surfaces, the vectors are points in 3-space and returned [x0,y0,z0,x1,y1,z1,...].
- For rational surfaces each vector contains a point in 3-space followed by a weight for the point. The points are returned [x0,y0,z0,w0,x1,y1,z1,w1,...]. The weights are positive.

Number of columns of control points 'ncol':

- For surfaces with non-periodic rows 'ncol' >= 'uorder'.
- For surfaces with periodic rows 'ncol' >= 3.

Number of rows of control points 'nrow':

- For surfaces with non-periodic columns 'nrow' >= 'vorder'.
- For surfaces with periodic columns 'nrow' >= 3.

Knot vector in the u direction 'uknots':

- The knot values form a non-decreasing sequence.
- If the rows are not periodic there are ('ncol' + 'uorder') knot values.
- If the rows are periodic there are ('ncol' + 1) knot values.
- The knot values follow the rules described in CRBSPS, see page .

Knot vector in the v direction 'vknots':

- The knot values form a non-decreasing sequence.
- If the columns are not periodic there are ('nrow' + 'vorder') knot values.
- If the columns are periodic there are ('nrow' + 1) knot values.
- The knot values follow the rules described in CRBSPS, see page .

Properties list 'props': The following tokens may be returned:

- PAPRPU - the surface is periodic in u (i.e. the rows are periodic).
- PAPRPV - the surface is periodic in v (i.e. the columns are periodic).

  The default is that the surface is not periodic.

Number of properties 'nprops':

- Gives the number of properties in the 'props' array.

Surface sense 'sense'

- If 'sense' is KI_true the surface normal is given by the cross product of the first derivatives of the surface with respect the u and v parameters. i.e. normal = Pu X Pv
- If 'sense' is KI_false the surface normal is in opposite to that described for 'sense' == KI_true.

OUBSSU - Outputs a region of a surface in B-Spline form

Receives

```
KI_tag_surface     *surf          surface
KI_dbl              urange[2]     urange of surface
KI_dbl              vrange[2]     vrange of surface
KI_dbl_distance    *tol           tolerance for approximations
<KI_int_nitems>    *nopts         number of options in 'iopts'
KI_cod_srop         iopts[nopts]  conversion options
```

Returns

```
KI_tag_list_dbl    *ctrl          control points
KI_int_dimension   *dim           dimension of control points
KI_int_order       *uorder        order of surface patches in u
KI_int_order       *vorder        order of surface patches in v
KI_int_nitems      *ncol          number of cols of control points
KI_int_nitems      *nrow          number of rows of control points
KI_tag_list_dbl    *uknots        knot vector in the u direction
KI_tag_list_dbl    *vknots        knot vector in the v direction
<KI_tag_list_int>  *props         list of surface properties
<KI_int_nitems>    *nprops        number of surface properties
KI_cod_error       *ifail         failure code
```

Specific Errors

```
KI_tolerances_too_tight        tolerance too tight
KI_cant_make_bspline           failed to convert geometry
```

Description The function outputs a region of a surface in B-Spline form.

The B-spline will represent the surface exactly if possible, but it may be necessary to approximate it. If so, 'tol' specifies an upper bound on the distance between the B-spline and the original surface, which this function will usually satisfy although this is not guaranteed.

Representations of planes and the rational forms of cylinders, cones, spheres and tori are exact. When the representation of a B-surface is changed it will, in general, only be exact when its order is not decreased.

The parameter limits of the new surface are defined using 'urange' and 'vrange'. For each pair of parameter limits, the following rules apply:

- The first element must be less than the second.
- Both elements must lie inside the parameter range, as given by ENSUPA, unless the corresponding parameter is periodic. In that case the first must lie in the range, and the difference between the two may not exceed the period.

The format of the output surface is selected by options in 'iopts'. Options allowed are:-

Option	Description
SROPCU	Force output of B-splines of degree 3
SROPNR	Force output of non-rational B-splines

The effects of these options on the representations produced for different surface types are:-

Planes

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	2	4	4	2
`vorder'	2	4	4	2
`dim'	3	3	3	3

Cylinders and Cones

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	4	4	4	4
`vorder'	2	4	4	2
`dim'	4	4	3	3

Spheres and Tori

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	4	4	4	4
`vorder'	4	4	4	4
`dim'	4	4	3	3

B-surfaces

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	*	4	4	* or 4
`vorder'	*	4	4	* or 4
`dim'	*	* or 3	3	3

Where * fields are taken from surface definition. The value of "* or n" fields is taken from the surface definition if the representation is exact, otherwise the value n is used.

Swept surfaces

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	*	4	4	*
`vorder'	2	4	4	2
`dim'	*	* or 3	3	3

Where * fields are derived from the curve which was swept. The value of "* or n" fields is derived from the swept curve if the representation is exact, otherwise the value n is used.

Spun surfaces

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	*	4	4	4
`vorder'	4	4	4	4
`dim'	4	4	3	3

Where the * field is derived from the curve which was spun.

Offset surfaces

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	*	4	4	*
`vorder'	*	4	4	*
`dim'	*	*	3	3

Where * fields are the same as for the surface which was offset.

All other surfaces

	none	SROPCU	SROPCU SROPNR	SROPNR
`uorder'	4	4	4	4
`vorder'	4	4	4	4
`dim'	3	3	3	3

The meaning of the values in the arguments 'ctrl', 'dim', 'uorder', 'vorder' 'ncol', 'nrow', 'uknots' and 'vknots' are given under OUBSPS.

Properties list 'props': The following tokens may be returned:

- PAPRPU - the B-spline surface is periodic in u
- PAPRPV - the B-spline surface is periodic in v. The default is that the B-spline surface is not periodic.
- PAPREX - the B-spline surface represents the original surface exactly. If this token is not present then the B-spline approximates the original.

Number of properties 'nprops':

- Gives the number of properties in the 'props' list.

OUCOCU - Output coordinates on curve

Receives

KI_tag_curve	*curve	curve to be output
KI_vec_position	start	starts from here
KI_vec_position	end	ends here
double	*ctol	max permitted chordal error
double	*atol	max permitted angular error
double	*stol	max permitted step length

Returns

<KI_tag_list_dbl>	*posns	list of vector components
KI_int_nitems	*npos	number of items in list
KI_cod_error	*ifail	failure code

Specific Errors

KI_tolerances_too_tight	failed to meet tolerances
KI_stol_too_small	invalid step length control set
KI_atol_too_small	invalid angular control set
KI_ctol_too_small	invalid chordal control set
KI_wrong_direction	start and end in wrong order
KI_coincident	start and end coincide but curve not closed
KI_not_on_curve	coords not within resolution distance of curve

Description This routine outputs coordinates along 'curve' from 'start' to 'end'. The user may specify the controls that govern the output of the coordinates or request that coordinates be output to default controls.

To help explain the meaning of 'ctol', 'atol', and 'stol', a chord is defined to be the straight line between adjacent coordinates. A step is the process of progressing from one location to the next.

'ctol' is the max permitted chordal error. That is the maximum permitted distance from a chord to the curve between the ends of the chord.

'atol' is the max permitted angular error. It is the maximum permitted sum of the angles between the chord and the tangents to the curve at the ends of the chord.

'stol' is the max permitted chordal length.

'ctol', 'atol', and 'stol' are independent means of controlling the output of the points. There will be no step where a valid 'ctol', 'atol', or 'stol' limit is exceeded. There is no guarantee that 'ctol', 'atol' or 'stol' will actually be attained.

There are lower limits to the values of 'ctol', 'atol' and 'stol' that can be used as controls. These values are not specified here as they depend upon the model resolution. Where these limits are exceeded suitable failure codes are returned.

If all the controls are set to <= 0.0, default controls are applied which capture the shape of the curve. Where some of the controls are set to <= 0.0, those controls that are <= 0.0 are ignored and coordinates are output to the remaining valid controls.

The positions stepped to are returned in 'posns', a list of doubles in the form

Xn, Yn, Zn, X(n+1), Y(n+1), Z(n+1), X(n+2) ......

where n varies from 0 to 'npos'/3 - 1.

X0,Y0,Z0 is 'start' first location stepped to and

XL,YL,ZL is 'end' last location stepped to.

where L = 'npos'/3 - 1

'start' to 'end' must follow the natural direction of the curve as indicated in the appropriate KI curve creation routine. If the curve is a trimmed curve the bounds are ignored. However valid vectors should be supplied.

OUCPCU - Outputs a constant parameter line curve

Receives

```
KI_tag_curve       *curve   Tag of constant parameter curve
```

Returns

```
KI_tag_surface     *surf    Underlying surface
KI_cod_papr        *uorv    Constant u or constant v
KI_dbl             *param   Parameter value
KI_cod_logical     *sense   Curve sense
KI_cod_error       *ifail   Failure indicator
```

Specific Errors

```
KI_bad_type                    Invalid curve.
```

Description This function, given the tag of a constant parameter line curve, returns the information which defines the curve.

'curve' is the existing curve from which the information is to be obtained.

'surf' is the tag of the underlying surface on which the curve lies.

'uorv' indicates whether the curve is :

Constant u (i.e. in the v direction) (uorv = PAPRUP)

Constant v (i.e. in the u direction) (uorv = PAPRVP)

'param' gives the constant parameter value.

'sense' is the sense of the constant parameter curve.

OUEXSU - Output extruded surface

Receives

```
KI_tag_surface       *surf     extruded surface to output
```

Returns

```
KI_tag_curve         *profil   curve extruded
KI_vec_direction      path     direction of sweep
KI_cod_logical       *sense    surface sense
KI_cod_error         *ifail    error code
```

Specific Errors

```
KI_unsuitable_entity Entity is not a surface of linear extrusion
```

Description The details of a surface of linear extrusion (type TYSUSE) are returned.

OUFEAT - Output items in feature

Receives

```
KI_tag_feature         *featre   feature
```

Returns

```
KI_cod_tyfe            *fetype   type-code of feature
KI_tag_list_entity     *entys    list of entities in feature
<KI_int_nitems>        *nentys   number of items in 'entys'
KI_cod_error           *ifail    failure code
```

Description The items in the feature are returned in the list 'entys'. If the feature is empty 'entys' is returned as an empty list and 'nentys' as 0.

OUFEAT outputs all the entities in the feature irrespective of their type. The entities of a particular type may be obtained using IDCOEN.

Can be called from the GO.

OUFGCU - Outputs a Foreign Geometry (FG) curve

Receives

```
KI_tag_curve          *curve          Foreign curve.
KI_int_nchars         *arrlen         Length of input key array.
```

Returns

```
KI_chr_key             key[arrlen]    Curve key.
KI_int_nchars         *keylen         Length of curve key.
<KI_tag_list_int>     *ivals          List of integer values.
<KI_tag_list_dbl>     *rvals          List of real values.
KI_cod_logical        *sense          Curve sense.
<KI_tag_transform>    *tf             Curve transformation.
KI_cod_error          *ifail          Failure indicator
```

Specific Errors

```
KI_unsuitable_entity       not a foreign geometry curve
```

Description This function, given a tag, outputs the data of a foreign curve.

'curve' - The tag of a valid foreign curve.

'arrlen' - The length of the key array being passed to this KI function.

'key' - The array of characters into which the key of the curve is copied. (If the array is not long enough to store the key, then the key is truncated.)

'keylen' - The true length of the curve's key.

'ivals' - Integer values supplied to the foreign curve evaluator.

'rvals' - Real values supplied to the foreign curve evaluator.

'sense' - The curve's sense.

'tf' - The tag of the curve's transformation.

OUFGSU - Outputs a Foreign Geometry (FG) surface

Receives

```
KI_tag_surface      *surf         Foreign surface.
KI_int_nchars       *arrlen       Length of input key array.
```

Returns

```
KI_chr_key           key[arrlen]  Surface key (truncated if necessary)
KI_int_nchars       *keylen       True length of surface key.
<KI_tag_list_int>   *ivals        List of integer values.
<KI_tag_list_dbl>   *rvals        List of real values.
KI_cod_logical      *sense        Surface sense.
<KI_tag_transform>  *tf           Surface transformation.
KI_cod_error        *ifail        Failure indicator
```

Specific Errors

```
KI_unsuitable_entity       not a foreign geometry surface
```

Description This function, given a tag, outputs the data of a foreign surface.

'surf' - The tag of a valid foreign surface.

'arrlen' - The length of the key array being passed to this KI function.

'key' - The array of characters into which the key of the surface is copied. (If the array is not long enough to store the key, then the key is truncated.)

'keylen' - The true length of the surface key.

'ivals' - Integer values supplied to the surface evaluator.

'rvals' - Real values supplied to the surface evaluator.

'sense' - The surface sense.

'tf' - The tag of the surface transformation.

In cases where the foreign surface was created without a transform and only identity transforms have been subsequently applied to it, the returned transformation tag 'tf' will be the NULTAG (APPTRA does not modify the transformation when identity transformations are applied).

OUFINF - Output information about the specified file

Receives

```
KI_int_nchars     *nchars          number of characters in filename
KI_chr_filename    filnam[nchars]  filename
int               *guise           what sort of file it is
int               *format          text or binary
KI_cod_slfi       *selcod          selection code specifying what
                                   information is wanted
```

Returns

```
int               *ival            returned information (integer)
double            *rval            not yet used
char               string[nstrng]  not yet used
<KI_int_nchars>   *nstrng          not yet used
KI_cod_error      *ifail           error code
```

Specific Errors

```
KI_corrupt_file          file header not as expected, bad header to file
KI_file_access_error     unexpected file access error
KI_cant_find_file        cannot find file
KI_bad_filename          invalid filename
KI_wrong_format          wrong format for specified guise
KI_file_read_corruption  corrupt data read, perhaps an NFS problem
KI_schema_access_error   file referred to non existent schema
KI_cant_open_file        cannot open file
KI_bad_file_format       invalid value given for format
KI_bad_file_guise        invalid value given for guise
```

Description Currently, the only acceptable value for 'selcod' is SLFIVN (SLFI_VersioN), which causes OUFINF to return the modeler version number under which the file was generated: this is returned as an integer value in 'ival'.

'nchars', 'filnam', 'guise', 'format' should correspond to the parameters expected by the Frustrum function FFOPRD

The following table gives the appropriate 'guise' and 'format' for the types of file for which OUFINF can be used :-

	guise	**format**
C-text-snapshot file	FFCSNP	FFTEXT
C-binary-snapshot file	FFCSNP	FFBNRY
C-journal file	FFCJNL	FFTEXT
C-binary-transmit file	FFCXMT	FFBNRY
C-text-transmit file	FFCXMT	FFTEXT
Fortran-binary-transmit file	FFCXMO	FFBNRY
Fortran-text-transmit file	FFCXMO	FFTEXT
C-binary-Schema files	FFCSCH	FFBNRY
C-text-Schema files	FFCSCH	FFTEXT

Note: OUFINF does NOT verify that the contents of the file are correct, although it will return error KI_corrupt_file if the header information is not as expected. This error will also arise if the file does not match the guise and format specified; for example, if you try to open a Fortran transmit file using the guise for a C-transmit file.

OUGEEF - Output geometry of edge or fin

Receives

```
KI_tag_topology *ed_fn   edge or fin being queried
KI_cod_logica   *parms   request for parameters
```

Returns

```
KI_tag_curve    *curve   tag of curve
KI_cod_tycu     *cutype  type of curve
KI_vec_position  start   start position
KI_dbl          *st_t    parameter of 'start'
```

```
KI_vec_position  end     end position
KI_dbl          *end_t   parameter of 'end'
KI_cod_logical  *sense   true if 'curve' is in same direction as 'ed_fn'
KI_cod_error    *ifail   failure code
```

Specific Errors

```
KI_missing_geom   Insufficient geometry to represent 'ed_fn'
KI_wrong_entity   Wrong entity
```

Description This function receives an item of topology, 'ed_fn', which may be an edge or a fin, and returns geometry representing it. The geometry comprises the tag of a curve, the curve type, position vectors on the curve corresponding to the end points of the item, and optionally the curve parameters of these position vectors. Together these represent a bounded portion of curve corresponding to 'ed_fn'.

The position vectors 'start' and 'end' are returned such that movement from 'start' to 'end' along the curve is in the same direction as the curve. If 'sense' is returned as KI_false, then the curve direction is in the opposite direction to 'ed_fn', though this can only occur for fins and laminar edges on sheet bodies.

Note that 'curve' is attached to the model and should not be modified or deleted without due care.

If 'parms' is set to KI_true, the curve parameters corresponding to 'start' and 'end', ('st_t' and 'end_t' respectively) will be calculated. 'st_t' will always be smaller than 'end_t', the period of parametrisation being added to 'end_t' if necessary. If 'parms' is set to KI_false, the parameters will be arbitrarily returned as zero.

If 'ed_fn' has no curve attached, an attempt will be made to return equivalent geometry that lies in the same surface. Thus if 'ed_fn' is an edge, fin geometry may be returned, and vice versa. However if 'ed_fn' is a fin, the geometry of the other fin will not be returned.

If 'ed_fn' is a ring, 'start' and 'end' will both be returned as the position vector corresponding to the lowest value in the parameter range for the curve. 'st_t' will be this lowest parameter value, and 'end_t' this value plus the period of parametrisation.

Where no geometry representing 'ed_fn' can be found, this will give rise to ifail KI_missing_geom.

Can be called from the GO.

OUGESU - Output generated surface

Receives

```
KI_tag_surface          *surfac  surface to output
```

Returns

```
KI_tag_list_int         *sftype  type and subtypes of surface
KI_int_nitems           *ntypes  number of entries in 'sftype'
KI_tag_list_int         *codes   codes formatting return data
KI_int_nitems           *ncode   number of format codes
<KI_tag_list_int>       *ints    integer data
<KI_tag_list_dbl>       *reals   real data
<KI_tag_list_geometry>  *geoms   underlying geometries
<KI_tag_list_curve>     *singc   list of singular curves
```

```
<KI_int_nitems>          *nsingc  number of singular curves
<KI_tag_list_dbl>        *singp   singularities
<KI_tag_list_curve>      *singo   owners of singularities
<KI_int_nitems>          *nsingp  number of singularities
KI_cod_logical           *sense   surface sense
KI_cod_error             *ifail   failure indicator
```

Specific Errors

```
KI_wrong_sub_type   not a surface supported by this routine
```

Description This routine outputs a swept or spun surface. Information about the surface is returned in the form of several lists, some of which may be null for some surface types:

- a list of codes which enable the data from the integer, real and geometry lists to be interpreted
- lists of integers and reals giving specific details about the surface
- a list of underlying geometries, from which the surface is generated
- lists of singular curves and points

The surface type and the sense of the surface are also returned.

Each type of surface is output to a fixed format, so it is possible to interpret the integer, real and geometric data solely from a knowledge of the surface type. However, it may be necessary to read the codes to decide whether a particular piece of data has been output for a particular surface. Conversely, the data can be interpreted completely from an understanding of the various codes, without looking at the surface type.

The surface type, which is a token from the range TYSU00, is returned in the list 'sftype'. In version 2.0, only one element of the list is used, so 'ntypes' always has the value 1.

The surface may have point or curve singularities (creases) and these are returned in separate lists. The surface normal and curvature cannot be calculated at either type of singularity. An example of a point singularity occurs when a swung curve meets the swing axis. A curve singularity is formed when a curve with a singularity (such as an intersection curve that meets itself at a terminator) is swept or swung. Curve singularities are returned as tags of curves in the list 'singc'; 'nsingc' is the number of singular curves. Point singularities are returned in the list 'nsingp' and the number of singular points is 'nsingp'. The argument 'singo' (owners of singular points) is not used at present, and is always null.

Each surface has a default normal direction, but this may be reversed. The 'sense' argument is set to KI_true if the surface normal is in the default direction, KI_false otherwise. The default direction varies according to the surface type, and is described in the information for each surface below.

Data returned for Swept Surface

A swept surface is one that is generated by sweeping a planar curve in a straight line normal to the plane. The real and geometry lists are used to output data defining a swept surface. The real list contains the sweep direction, and the curve that is swept to form the surface (the section curve) is in the geometry list. If the curve has singularities, then these will be swept into singular curves (which will actually be straight lines).

'sftype':	TYSUSE
'sbtype':	-

'ncode': 2

'nsingc': corresponds to number of singularities on section curve

'nsingp': 0

length of 'ints' list: 0

length of 'reals' list: 3

length of 'geoms' list: 1

default normal direction: cross product of section curve tangent and sweep direction

`codes'	list	data	start	length
OUFODR	`reals'	sweep direction	1	3
OUFOCU	`geoms'	curve to be swept	1	1

Data returned for Swung Surface

A swung surface is generated by swinging a planar curve about an axis in its plane. The real and geometry lists are used to output data defining a swung surface. The real list contains the swing axis, and the curve that is swung to form the surface (the profile curve) is in the geometry list. If the curve has singularities, then these will be swung into singular curves (which will actually be circles). There may also be point singularities where the profile curve meets the axis. The profile may be bounded by two end points; this will be necessary if it intersects the axis. Note that the tokens in 'codes' are used to indicate whether bounds are present in the 'reals' list, but that the 'reals' list will always have the same length whether or not bounds are present.

'sftype': TYSUSU

'sbtype': -

'ncode': 4

'nsingc': corresponds to number of singularities on profile curve

'nsingp': 0, 1 or 2

length of 'ints' list: 0

length of 'reals' list: 12

length of 'geoms' list: 1

default normal direction: cross product of the profile curve tangent and the swing direction;

the swing direction is clockwise when viewed down the swing axis

`codes'	list	data	start	length
OUFOAX	`reals'	swing axis	1	6
OUFOPV	`reals'	start position of valid region on curve	7	3
OUFONP	`reals'	no start position	7	3

OUFOPV	`reals'	end position of valid region on curve	10	3
OUFONP	`reals'	no end position	10	3
OUFOCU	`geoms'	curve to be swung	1	1

OUIDEN - Output identifier of entity

Receives

```
KI_tag_entity      *entity    entity whose id is required
```

Returns

```
KI_int_id          *id        id of specified entity
KI_cod_error       *ifail     error code
```

Description Returns the identifier 'id' of entity 'entity'.

Any entity which is attached to a part has an identifier, which is unique within that part. This identifier is allocated automatically when the entity is attached to the part, and is maintained for as long as the entity is attached to the part; this applies even when the part is re-received into a later session. When an entity is transferred from one part to another, it will be allocated a new identifier appropriate to the destination part. In a boolean operation, for example, all entities within the target body will retain their identifiers, but those in the tool body will be allocated new identifiers appropriate to the target body.

The identifier of an entity which is not attached to a part is returned as zero.

Can be called from the GO.

OUIDLS - Output identifiers of list of entities

Receives

```
KI_tag_list_entity    *entys   entities whose ids are required
```

Returns

```
KI_tag_list_int       *ids     ids of specified entities
KI_int_nitems         *nids    number of entries in 'ids'
KI_cod_error          *ifail   error code
```

Description Returns list of identifiers ('ids') corresponding to the given list of entities ('entys'), such that the nth entry in 'ids' gives the identifier for the nth entry in 'entys'. 'nids' gives the length of list 'ids' (which is also the length of list 'entys').

The entities given in 'entys' need not all belong to the same part. The identifiers for any entities which are not attached to a part will be returned as zero.

See OUIDEN for explanation about identifiers.

Can be called from the GO.

OUINTP - Output interface parameter

Receives

```
KI_cod_slip       *pnum     parameter code
```

Returns

```
int             *ival    integer value of parameter
double          *rval    real value of parameter
KI_cod_error    *ifail   failure indicator
```

Description Outputs the value of an interface parameter.

'pnum' specifies which parameter is to be returned: different parameters may be integer or real values and are returned in the appropriate return argument. At present all parameters returned from this routine are integers.

The valid values of 'pnum' and the corresponding parameters are:

`pnum'	Return	Significance
SLIPCH	`ival'	If non-zero, all KI routines will check that their parameters have acceptable values. If zero, KI routine may not check their parameters.
SLIPJO	`ival'	If non-zero, all KI routines will record their received and returned values in the journal file.
SLIPBB	`ival'	Flag indicating whether the Kernel maintains a bulletin board. The defined val ues of 'ival' are: 0 no bulletin board maintained 1 bulletin board is maintained for tags only 2 bulletin board is maintained for tags user fields
SLIPRB	`ival'	If non-zero the modeler will log all changes to the model. Only if this is done may rollback marks be set or used. The specific value of 'ival' is the size of the rollback file in bytes.
SLIPRF	`ival'	If 1 then ROLBFN can be used to roll forward. If 0 then roll forward will not be allowed but the amount of data stored in the rollback file will be reduced.
SLIPLC	`ival'	If non-zero, the Kernel will perform local checks on geometry and topology after a local modification is made to the model by CRFASU, CRSOFA, DELFAS, RMFASO, SWEENT, SWIENT, TWSUFA, TWEFAC, TAPFAS or BLEFIX.
SLIPDC	`ival'	If non-zero, limited checks on the consistency geometry attached to a model with ATTGEO an neighboring geometry will be made.
SLIPUF	`ival'	If non-zero user fields are received with archived parts. If SLIPUF is zero user-field values of received parts are set to zero.
SLIPGS	`ival'	If non-zero, generated surfaces can be created during a call to SWEENT or SWIENT. If SLIPGS is zero, B-surfaces will be created instead.

SLIPBT	`ival'	Shows the use of binary transmission. The defined values of 'ival' are: 1 text Receive, text Transmit 2 binary Receive, text Transmit 3 text Receive, binary Transmit 4 binary Receive, binary Transmit 5 text Receive, neutral Transmit 6 binary Receive, neutral Transmit 7 applio Receive, text Transmit 8 applio Receive, binary Transmit 9 applio Receive, neutral Transmit 10 applio Receive, applio Transmit 11 text Receive, applio Transmit 12 binary Receive, applio Transmit
SLIPSN	`ival'	Shows the use of binary snapshots. The defined values of 'ival' are: 1 text get snapshot, text save snapshot 2 binary get snapshot, text save snapshot 3 text get snapshot, binary save snapshot 4 binary get snapshot, binary save snapshot
SLIPSI	`ival'	If non-zero, the Kernel will perform self intersection checks on geometry during operations which require the geometry to pass the checks imposed by CHCKEN.
SLIPCO	`ival'	Flag indicating whether the Kernel will perform composite geometry checks on B-curves and B-surfaces when determining whether the geometry can be modeled as a single topological entity. The defined values of `ival are : 0 Perform all composite geometry checks 1 Perform no composite geometry checks Values outside this range return KI_bad_value. This flag is set to non-zero in STAMOD.

SLIPTL	`ival'	If non-zero, the modeler will check all tags allocated against an upper bound equal to the absolute value of 'ival'. An operation that would other wise result in the allocation of tags exceeding 'ival' will fail, returning KI_tag_limit_exceeded.
SLIPGT	`ival'	Flag indicating whether general bodies are to be legally returned by KI func tions - principally boolean operations. With SLIPGT set to zero, general bodies will not be returned and the operations will fail with the same ifail as at previous versions i.e. KI_non_manifol. With SLIPGT set to one, general bodies will be returned as valid results.

OULERR - Output information about the most recent KI-error

Receives

```
KI_cod_sler             *selcod        enquiry code
```

Returns

```
int              *ival            integer value
char              string[nchars]  null-terminated character string
<KI_int_nchars> *nchars          length of character string (excluding
                                  terminal null)
KI_cod_error    *ifail           error code
```

Description OULERR returns information about the most recent KI-error (ie non-zero ifail return from a KI routine), and is principally intended as a debugging aid to help pinpoint the precise cause of the error. The value of 'selcod' determines what information is returned :-

`selcod' value	interpretation of `selcod' value	`ival' return value	`string' return value	max-length of string
SLERRO	SLER_ROutine	Zero	KI-routine-name	32
SLEREC	SLER_Error_Code	ifail-value	ifail-mnemonic	32
SLEREX	SLER_EXplana tion	ifail-value	explanation of error code	256
SLERAR	SLER_ARgument	argument number ([-1],1,2,...)	name of faulty argument	32
SLERAI	SLER_Array_Index	array-index ([-1],0,1,...)	null	1
SLERLE	SLER_List_Entry	list-entry no ([-1],1,2,...)	null	1
SLERTG	SLER_TaG	tag value [or -1]	null	1

For codes SLERAR, SLERAI, SLERLE, SLERTG, 'ival' may be returned as (-1) to indicate that the enquiry is not relevant to this error (for example, the error may not be specific to a particular argument).

Where 'string' is null, it will simply contain the terminal null, and the value of 'nchars' will be zero.

Can be called from the GO.

OUMODP - Output modeller parameter

Receives

```
KI_cod_slmp        *pnum     parameter code
```

Returns

```
int                *ival     integer value of parameter
double             *rval     real value of parameter
KI_cod_error       *ifail    failure indicator
```

Description Outputs the value of a modeller parameter.

'pnum' specifies which parameter is to be returned: different parameters may be integer or real values and are returned in the appropriate return argument. At present all parameters returned from this routine are reals.

The valid values of 'pnum' and the corresponding parameters are:

`pnum'	Return	Significance
SLMPLP	`rval'	Absolute (linear) precision of modeler as a real value greater than zero. Lengths and distances differing by no more than this value are treated as equal.
SLMPAP	`rval'	Angular precision of modeler as a real value greater than zero. Angles, and values calculated from normalized vectors, that differ by no more than this value are treated as equal.

OUOFSU - Output offset surface

Receives

```
KI_tag_surface          *offset      offset surface
```

Returns

```
KI_tag_surface          *under       underlying surface
KI_dbl                  *dist        offset distance
KI_cod_logical          *sense       surface sense
KI_cod_error            *ifail       failure code
```

Specific Errors

```
KI_unsuitable_entity      Entity is not an offset surface
```

Description The details of an offset surface (type TYSUOF) are returned.

OUPART - Output key and state of part

Receives

```
KI_tag_part           *part             part to output
KI_int_nchars         *buflen           length of key array
```

Returns

```
<KI_int_nchars>       *keylen           length of key
KI_chr_key             key[keylen]      key of part
KI_cod_enst           *state            state of part
KI_cod_error          *ifail            failure indicator
```

Specific Errors

```
KI_buffer_overflow            key too long for array
```

Description OUPART returns the key (if any) and state of 'part'.

The state of the part will be one of:

- new (ENSTNW) - the part has been created during this session or was anonymous and has been changed.
- stored (ENSTST) - the part has been transmitted to external storage or has been received from the archive and has not been changed.
- anonymous (ENSTAN) - the part is an unchanged version of a part transmitted as an un-keyed sub-part into the archive.
- modified (ENSTMD) - the part is a changed version of a part in the archive.
- unloaded (ENSTUN) - the part has not been received from the archive but is instanced by another part in memory or the part has been explicitly unloaded from memory with UNLDPA.

All stored, modified and unloaded parts have a key. New and anonymous parts do not have a key and zero will be returned in 'keylen'.

OUPWPC - Output B-curve in piecewise form

Receives

```
KI_tag_b_curve      *bc       B-curve
KI_cod_slba         *basis    representation method
```

Returns

```
KI_tag_list_dbl     *coeffs   vectors defining the curve
KI_int_dimension    *dim      dimension of defining vectors
KI_int_order        *order    order of curve
KI_int_nitems       *nseg     number of segments in curve
KI_cod_error        *ifail    failure indicator
```

Specific Errors

```
KI_bad_order     order must be four for Hermite basis
```

Description This function outputs a B-curve in a piecewise form chosen by the user. The following methods of representing the data are available:

- Bezier ('basis' = SLBABZ)
- Polynomial ('basis' = SLBAPY)
- Hermite (cubic only) ('basis' = SLBAHE)
- Taylor series ('basis' = SLBATA)

Coefficient data 'coeffs':

- Contains 'order'*'nseg' vectors of dimension 'dim'. If 'dim'=3, then the vectors are 3-D vectors containing the x, y and z components. If 'dim'=4, then each vector has a weight (w) associated with it, and x, y, z and w components are returned for each vector.
- The coefficients are returned in order, segment by segment.
- The interpretation of the coefficients depends on the representation method chosen; this is determined by the value of the argument 'basis'.

The expressions for each segment of the B-curve P(t) in the various representations are given below. For generality, the rational form is given. The simplification to the non-rational form can be obtained by setting the weights equal.

- Bezier vertices SLBABZ:

  The equation of a rational Bezier curve segment is:

$$P(t) = \frac{\sum_{i=0}^{n} b_i(t) w_i V_i}{\sum_{i=0}^{n} b_i(t) w_i}$$

  Where:

  $n = \text{'order'}-1$

  $V_i = \text{Bezier vertex}$

  $w_i = \text{weight for } V_i$

  $b_i(t) = \text{Bezier coefficients, defined by:}$

$$b_i(t) = \frac{n!}{i!(n-i)!} * t^i * (1-t)^{n-i}$$

The Bezier vertices are returned V_0 ,w_0 ,...,V_n ,w_n for the rational form, or V_0,...,V_n for the non-rational form.

- Polynomial coefficients SLBAPY:

  The curve equation is given by a rational polynomial of order 'order':

  Where:

  $n = \text{'order'}-1$

  $A_i = \text{Polynomial coefficient}$

  $w_i = \text{weight for } A_i$

  The polynomial coefficients are returned starting with the constant term and ending with the term of highest degree.

- Hermite coefficients SLBAHE:

$$P(t) = \frac{\sum_{i=0}^{n} w_i A_i t^i}{\sum_{i=0}^{n} wit^i}$$

This method can only be used for cubics. The equation of the curve is:

$$P(t) = \frac{f0(t)\;\; w0\;\; P0 + g0(t)\;\; w1\;\; P1 + f1(t)\;\; d0\;\; D0 + g1(t)\;\; d1\;\; D1}{f0(t)\;\; w0 + g0(t)\;\; w1 + f1(t)\;\; d0 + g1(t)\;\; d1}$$

Where:

$f0(t) = 1 - 3t^2 + 2t^3$ $\quad$ $g0(t) = 3t^2 - 2t^3$

$f1(t) = t - 2t^2 + t^3$ $\quad$ $g1(t) = -t^2 + t^3$

$P0, P1 =$ start and end points of segment

$D0, D1 =$ derivatives at start and end

$w0, w1 =$ weights at end points

$d0, d1 =$ derivatives of weights at start and end

The coefficients are returned as P0, w0, P1, w1, D0, d0, D1, d1 for the rational form, or P0, P1, D0, D1 for the non-rational form.

- Taylor series SLBATA:

  This method stores the derivatives evaluated at the point start of each segment, allowing the curve to be reconstructed as a Taylor series:

  Where:

  $n =$ ‘order‘-1

  $P^{(i)} =$ i‘th derivative at t=0

  $w^{(i)} =$ i‘th derivative of weight at t=0

  The point is returned first, followed by the 1st derivative and ending with the derivative of order 'order'-1.

Dimension of coefficient vectors 'dim':

- For rational curves 'dim'=4.
- For non-rational curves 'dim'=3.

$$P(t) = \frac{\sum_{i=0}^{n} \frac{w^{(i)} P^{(i)} t^i}{i!}}{\sum_{i=0}^{n} \frac{w^{(i)} t^i}{i!}}$$

Order of each segment of the curve 'order':

- The order of the curve = degree + 1.
- The minimum order is 2.
- The Hermite basis ('basis' = SLBAHE) may only be chosen if the curve is a cubic ('order' = 4).

OUPWPS - Output B-surface in piecewise form

Receives

```
KI_tag_b_surface  *bs          B-surface
KI_cod_slba       *basis       representation method
```

Returns

```
KI_tag_list_dbl   *coeffs      vectors defining the surface
KI_int_dimension  *dim         dimension of defining vectors
KI_int_order      *uorder      order of surface in u
KI_int_order      *vorder      order of surface in v
KI_int_nitems     *ncol        number of columns of patches
KI_int_nitems     *nrow        number of rows of patches
KI_cod_logical    *sense       surface sense
KI_cod_error      *ifail       failure indicator
```

Specific Errors

```
KI_bad_order     order must be four for Hermite basis
```

Description This function outputs a B-surface in a piecewise form chosen by the user. The following methods of representing the data are available:

- Bezier ('basis' = SLBABZ)
- Polynomial ('basis' = SLBAPY)
- Hermite (bicubic only) ('basis' = SLBAHE)
- Taylor series ('basis' = SLBATA)

Coefficient data 'coeffs':

- Contains ('uorder' * 'vorder' * 'ncol' * 'nrow') vectors of dimension 'dim'. If 'dim'=3, then the vectors are 3-D vectors giving the x, y and z components. If 'dim'=4, then each vector has a weight (w) associated with it, and x, y, z and w components are returned for each vector.

- The data is returned patch by patch, row by row.
- The interpretation of the patch data depends on the representation method chosen; this is determined by the value of the argument 'basis'.

The expressions for each patch of the B-surface P(u,v) in the various representations are given below. For generality, the rational form is given. The simplification to the non-rational form can be obtained by setting the weights equal.

- Bezier vertices SLBABZ:

  The equation of a rational Bezier surface patch is:

$$P(u, v) = \frac{\sum_{i=0}^{nu} \sum_{j=0}^{nv} b_i(u) b_j(v) w_{ij} V_{ij}}{\sum_{i=0} \sum_{i=0} b_i(u) b_j(v) w_{ij}}$$

  Where:

  $nu = \text{`uorder`-1}$

  $nv = \text{`vorder`-1}$

  $V_{ij} = \text{Bezier vertex}$

  $w_{ij} = \text{weight for } V_{ij}$

  $b_i(u), b(v)_j = \text{Bezier coefficients}$

  For the rational form the Bezier vertices and weights are returned:

  $V_{00}, w_{00}, V_{10}, w_{10}, \ldots, V_{mo}, w_{m0}, V_{01}, w_{01}, \ldots, V_{m1}, w_{m1}, \ldots, V_{0n}, w_{0n}, \ldots, V_{mn}, w_{m}$

  For the non-rational form the w's are missed out.

- Polynomial coefficients SLBAPY:

  The surface equation is given by a rational bipolynomial of orders 'uorder', 'vorder':

  Where:

  $nu = \text{`uorder`-1}$

  $nv = \text{`vorder`-1}$

  For the rational form the polynomial coefficients Aij are supplied:

  $A_{00}, w_{00}, A_{10}, w_{10}, \ldots, A_{mo}, w_{m0}, A_{01}, w_{01}, \ldots, A_{m1}, w_{m1}, \ldots, A_{0n}, w_{0n}, \ldots, A_{mn}, w_{mn}$

  starting with the constant term and ending with the term of highest degree. For the non-rational form the w's are missed out.

- Hermite coefficients SLBAHE

$$P(u, v) = \frac{\sum_{i=0}^{nu} \sum_{j=0}^{nv} w_{ij} A_{ij} u^i v^i}{\sum_{i=0}^{nu} \sum_{j=0}^{nv} w_{ij} u^i v^i}$$

This method can only be used for bicubics. The Hermite equation for the patch in matrix form is:

$$P(u, v) = \frac{(1uu^2u^3)MAM^T(1vv^2v^3)^T}{(1uu^2u^3)MWM^T(1vv^2v^3)^T}$$

where M = (1 0 0 0)
(0 0 1 0)
(-3 3 -2 -1)
(2 -2 1 1)

A = (w00*P00 w01*P01 wv00*Pv00 wv01*Pv01)
(w10*P10 w11*P11 wv10*Pv10 wv11*Pv11)
(wu00*Pu00 wu01*Pu01 wuv00*Puv00 wuv01*Puv01)
(wu10*Pu10 wu11*Pu11 wuv10*Puv10 wuv11*Puv11)

W = (w00 w01 wv00 wv01)
(w10 w11 wv10 wv11)
(wu00 wu01 wuv00 wuv01)
(wu10 wu11 wuv10 wuv11)

and the superscript T denotes the transpose.

In the matrices A and W, the coefficients P, Pu, Pv and Puv are the points at the corners and their derivatives. The w's are the corresponding weights and their derivatives. P00 denotes P(0,0), etc.

For the rational form the coefficients are returned:

P00, w00, P10, w10, P01, w01, P11, w11
Pu00, wu00, Pu10, wu10, Pu01, wu01, Pu11, wu11

Pv00, wv00, Pv10, wv10, Pv01, wv01, Pv11, wv11

Puv00, wuv00, Puv10, wuv10, Puv01, wuv01, Puv11, wuw11

For the non-rational form, the w's are missed out.

- Taylor series SLBATA:

  This method stores the derivatives evaluated u=0, v=0 corner of each patch, allowing the surface to be reconstructed as a Taylor series:

$$P(u, v) = \frac{\sum_{i=0}^{nu}\sum_{j=0}^{nv}\frac{w^{(i)(j)}P^{(i)(j)}u^i v^j}{i!j!}}{\sum_{i=0}^{nu}\sum_{j=0}^{nv}\frac{w^{(i)(j)}u^i v^j}{i!j!}}$$

Where:

$$nu = \text{`uorder`} - 1$$

$$nv = \text{`vorder`} - 1$$

$$P^{(i)(j)} = \frac{d^{i+j}P}{du^i dv^j}(0, 0)$$

$$w^{(i)(j)} = \frac{d^{i+j}w}{du^i dv^j}(0, 0)$$

The point is returned first, followed by the u derivatives in order and ending with the derivative of order 'uorder'-1 in u, 'vorder'-1 in v.

Dimension of coefficient vectors 'dim':

- For rational surfaces 'dim'=4.
- For non-rational surfaces 'dim'=3.

Order of each patch of the surface in u 'uorder', and in v, 'vorder':

- The order = degree + 1.
- The minimum order is 2.
- If the Hermite basis is chosen ('basis' = SLBAHE) then the surface has to be bicubic ('uorder' = 'vorder' = 4).

Surface sense 'sense'

- If 'sense' is KI_true the surface normal is given by the cross product of the first derivatives of the surface with respect the u and v parameters. i.e. normal = Pu X Pv
- If 'sense' is KI_false the surface normal is in opposite to that described for 'sense' == KI_true.

OURVSU - Output surface of revolution

Receives

```
KI_tag_surface        *surf     spun surface to output
```

Returns

```
KI_tag_curve          *profil   curve revolved
KI_vec_position        point    point on revolution axis
KI_vec_axis            direct   direction of revolution axis
KI_cod_logical        *sense    surface sense
<KI_int_nitems>       *nsings   number of singularities
<KI_tag_list_dbl>     *parms    parameter range on profile
KI_cod_error          *ifail    error code
```

Specific Errors

```
KI_unsuitable_entity   Entity is not a surface of revolution
```

Description The details of a surface of revolution (type TYSUSU) are returned.

The return argument 'nsings' is the number of singularities present on the surface and can be 0, 1, or 2. If 'nsings' is 1 or 2 then a list of two doubles is also returned which is the parameter range of the part of the profile curve which was used to create the surface.

OUSPCU - Output an SP-curve in B-spline form

Receives

```
KI_tag_curve         *spc       SP-curve
```

Returns

```
KI_tag_surface       *surf      basis surface of SP-curve
KI_tag_list_dbl      *ctrl      control points
int                  *dim       vertex dimension (2 or 3)
KI_int_order         *order     order of curve
KI_int_nitems        *nctrl     number of control points
KI_tag_list_dbl      *knots     knot values
KI_cod_logical       *period    periodic flag
KI_cod_error         *ifail     failure indicator
```

Specific Errors

```
KI_bad_type            non-sp curve supplied
```

Description OUSPCU will, given the tag of an SP-curve, output the SP-curve's data.

An SP-curve describes a 3-space curve in (2-dimensional) parameter space of a surface.

The SP-curve consists of general B-spline curves in surface parameter space (i.e. curves with (u,v) instead of (x,y,z) vertices, where u & v are the parameters of the surface). SP-curves can be rational, in which case 'dim' is 3, and the vertices are (u,v,w) vectors.

The SP-curve parametrisation is governed by the range of the knot vector 'knots'.

The following data is output:

- The surface ('surf') in whose parameter space the SP-curve has been computed.
- The B-spline control points 'ctrl'.
- The vertex dimension 'dim', which will be 2 for non-rational curves, and 3 for rational ones.
- The order of the SP-curve 'order'.
- The number of control points 'nctrl'.
- The knot vector 'knots'. There will be 'nctrl' + 'order' of these.
- If the B-spine is periodic, 'period' is set to KI_true.

Periodics: Parasolid may choose to treat an SP-curve as periodic if it is closed and G1 continuous even if the B-spline is not periodic and 'period' is KI_false, ENCUPA reflects this internal periodicity.

OUSPPC - Output B-curve as spline points

Receives

```
KI_tag_b_curve         *bc       B-curve
```

Returns

```
KI_tag_list_dbl        *pts      spline points
KI_int_nitems          *npts     number of spline points
KI_tag_list_int        *props    curve properties
KI_tag_list_<list>     *pdata    list of data lists
KI_int_nitems          *nprops   number of curve properties
KI_cod_error           *ifail    failure indicator
```

Specific Errors

```
KI_bad_knots          curve has unsuitable knot vector
KI_unsuitable_entit   linear or quadratic curve has more than one
y                     segemnt order of curve is greater than four
```

Description This function outputs a B-curve as spline points, end conditions and a knot vector. This constitutes sufficient information to recreate the curve (via CRSPPC).

Restrictions

- The curve must be cubic or lower degree, non-rational and have continuous first and second derivatives (note that for a linear or quadratic curve to satisfy these continuity requirements, it must consist of a single segment).
- If the curve is periodic it must not have any repeated knots.
- If the curve is non-periodic it may only have repeated knots at its ends.
- Any curve created by CRSPPC and not subsequently modified, will satisfy these conditions.

Spline points 'pts', 'npts':

The spline points are returned in the list of real data 'pts', in order along the curve. 'npts' points are returned. They correspond to ends of segments in the original curve.

Curve properties 'props', 'pdata', 'nprops':

The 'props' array contains 'nprops' tokens from the sequence PAPR00. A particular property may be associated with additional data; if so, this is returned in a list. The tag of the list is returned in the 'pdata' list,in the position corresponding to the token in 'props'.

The table shows which tokens may be present, and the data associated with them. Some tokens are always present, whereas others are only present if the curve has a particular property.

Token	Meaning	Real data	Always present?
PAPRPE	curve is periodic	none	NO - the default is non-peri odic
PAPRCS	derivative returned at start of curve (clamped end condition)	derivative vector	present unless curve is periodic
PAPRCE	derivative returned at end of curve (clamped end condition)	derivative vector	present unless curve is periodic
PAPRKT	knot vector	knot vector	YES
PAPRCU	the curve is cubic	none	NO

End conditions PAPRPE, PAPRCS, PAPRCE

Either clamped or periodic end conditions may be returned. Clamped end conditions refer to either the start or end of the curve, whereas the periodic end condition refers to both.

- Clamped end conditions return the first derivative, with respect to a parameter varying between 0 and 1 over the first or last interval, at the start and end of the curve. The derivative is returned in a real list of length 3.
- Periodic end conditions imply that the curve is closed, so that the curve returns to the start point after the final point has been splined. The curve meets itself with continuity of tangent and curvature.

Knot vector PAPRKT:

A knot vector is always returned. It has the following properties:

- The knot values form a strictly increasing sequence.
- If the curve is not periodic there will be 'npts' knot values.
- If the curve is periodic there will be ('npts'+1) knot values.

Cubic curve PAPRCU:

This token indicates that the curve is cubic.

OUSPPS - Output B-surface as spline points

Receives

```
KI_tag_b_surface       *bs       B-surface
```

Returns

```
KI_tag_list_dbl        *pts      mesh of spline points
KI_int_nitems          *ncol     number of columns of points
KI_int_nitems          *nrow     number of rows of points
KI_tag_list_int        *props    surface properties
```

```
KI_tag_list_<list>     *pdata    list of data lists
KI_int_nitems          *nprops   number of surface properties
KI_cod_logical         *sense    surface sense
KI_cod_error           *ifail    failure indicator
```

Specific Errors

```
KI_bad_knots           surface has unsuitable knot vector
KI_unsuitable_entity   more than one patch in linear or quadratic
                       direction order of surface is greater than four
```

Description This function outputs a B-surface as spline points, boundary conditions and knot vectors. This constitutes sufficient information to recreate the surface (via CRSPPS).

Restrictions

- The surface must be bicubic (or lower degree), non-rational and have continuous first and second derivatives. If a surface is linear or quadratic the u direction then to satisfy these conditions it must consist of a single column of patches (similarly for v direction and rows of patches).
- If the surface is periodic in a particular direction the corresponding knot vector must not contain any repeated knots.
- If the surface is non-periodic in a particular direction the corresponding knot vector can only contain repeated knots at the start and end.
- Any surface created using CRSPPS and not subsequently modified will satisfy these conditions.

Spline points 'pts', 'npts':

The spline points are returned in the list of real data 'pts', in order along the rows, row by row. 'ncol' * 'nrow' points are returned. They correspond to corners of the patches in the surface.

Surface properties 'props', 'pdata', 'nprops':

The 'props' array contains 'nprops' tokens from the sequence PAPR00. A particular property may be associated with additional data; if so, this is returned in a list. The tag of the list is returned in the 'pdata' list, in the position corresponding to the token in 'props'.

The table shows which tokens may be present, and the data associated with them. Some tokens are always present, whereas others are only present if the surface has a particular property.

In explaining the various properties of splined surfaces the following notation is used

- bottom boundary - first row of points
- top boundary - last row of points
- left boundary - first col of points
- right boundary - last col of points

Token	Meaning	Real data	Always present?
PAPRPU	rows are periodic	none	NO - the default is non-peri odic
PAPRPV	columns are periodic	none	NO - the default is non-peri odic

PAPRCB	derivative returned at start of columns (clamped boundary condition)	`ncol' derivative vectors	present unless the columns are periodic
PAPRCT	derivative returned at end of columns (clamped boundary condition)	`ncol' derivative vectors	present unless the columns are periodic
PAPRCL	derivative returned at start of rows (clamped boundary condition)	`nrow' derivative vectors	present unless the rows are periodic
PAPRCR	derivative returned at end of rows (clamped boundary condition)	`nrow' derivative vectors	present unless the rows are peri odic
PAPRBL	bottom left twist vector returned	twist vector	present unless the rows or columns are periodic
PAPRBR	bottom right twist vector returned	twist vector	present unless the rows or columns are periodic
PAPRTL	top left twist vector returned	twist vector	present unless the rows or columns are periodic
PAPRTR	top right twist vector returned	twist vector	present unless the rows or columns are periodic
PAPRKU	knot vector in u	knot vector	YES
PAPRKV	knot vector in v	knot vector	YES
PAPRCU	surface is bicubic	none	NO

Boundary conditions PAPRPU, PAPRPV, PAPRCB, PAPRCT, PAPRCL, PAPRCR:

Either clamped or periodic boundary conditions may be returned. These apply to u and v directions (rows and columns) independently. Clamped boundary conditions refer to either the start or end of the rows or columns. Periodic boundary conditions apply to both the start and end of rows or columns simultaneously.

- Clamped boundary conditions return the first derivatives, with respect to a parameter varying between 0 and 1 over the first or last interval, across some boundary of the surface. The derivatives are returned in a real list.
- Periodic boundary conditions imply that the rows or columns are closed, so that the surface meets itself with continuity of tangent and curvature.

Twist vectors PAPRBL, PAPRBR, PAPRTL, PAPRTR:

A twist vector is a derivative with respect to both u and v; i.e. it is the rate of change of the u derivatives in the v direction, and also the rate of change of the v derivatives in the u direction (these two quantities are always equal). The twist vectors are returned at all four corners, but only if neither the rows nor the columns are periodic.

Knot vector PAPRKU, PAPRKV:

Knot vectors are always returned. They have the following properties:

- The knot values form strictly increasing sequences.
- If the rows are not periodic there will be 'ncol' knot values in the u knot vector.
- If the rows are periodic there will be ('ncol'+1) knot values in the u knot vector.
- If the columns are not periodic there will be 'nrow' knot values in the v knot vector.
- If the columns are periodic there will be ('nrow'+1) knot values in the v knot vector.

Bicubic surface PAPRCU:

This token indicates that the surface is bicubic.

Surface sense 'sense'

- If 'sense' is KI_true the surface normal is given by the cross product of the first derivatives of the surface with respect the u and v parameters. i.e. normal = Pu X Pv
- If 'sense' is KI_false the surface normal is in opposite to that described for 'sense' == KI_true.

OUSTAT - Output information about the current state of the Kernel

Receives

```
KI_cod_slst  *selcod selection code specifying what information is
                     wanted
```

Returns

```
int          *ival   returned information (integer)
double       *rval   returned information (double)
KI_cod_error *ifail  error code
```

Description What information is returned depends on the value of 'selcod' :-

`selcod' value	Interpretation of `selcod' value	Return	Interpretation of return value
SLSTAR	SLST_At_Rollmark	`ival'	1 if at a rollmark, else 0
SLSTNF	SLST_Num_Forward	`ival'	number of roll-forward steps available
SLSTNB	SLST_Num_Back	`ival'	number of roll-back steps available (excluding rollback using ROLBLM)
SLSTVM	SLST_Virtual_Mem	`ival'	total bytes of virtual memory currently allo cated to the model data structure (including free space)
SLSTFS	SLST_Free_Space	`ival'	amount of free-space within the model data structure in bytes
SLSTMT	SLST_Maximum_Tag	`ival'	the highest tag value allocated by Parasolid

OUTATT - Output an attribute

Receives

```
KI_tag_attribute   *attrib        attribute
<KI_int_nitems>    *bufsiz        amount of space available in chars
```

Returns

```
KI_tag_entity     *owner          entity to which attribute is attached
<KI_tag_list_int> *ivals          list of integer values
<KI_tag_list_dbl> *rvals          list of real values
<KI_tag_list_int> *slens          list of string lengths
KI_chr_string      chars[bufsiz]  array of data for string fields
KI_cod_error      *ifail          error code
```

Description The owner and contents of an attribute are output. To find the type, and hence any other information dependent only on the type, use ENDFAT.

The integer fields are returned in the integer list ivals.

The real values for the fields of types real, axis, vector, coordinate and direction are returned in rvals, in the order of the fields (which may be obtained by calling OUATDF with the type of the attribute, obtained from ENDFAT). Thus the real values held in the first field of any of the above types appear first, followed by the real values held in the next field of any of these types. Each real field yields one real value, each axis six and each field of type coordinate, vector or direction three.

The string fields are returned, concatenated, in the array 'chars'; the lengths of the strings are returned in the list 'slens'. Thus if we denote the first entry in 'slens' by 'slens' and so on, the first string field is given by the first 'slens' values in 'chars', the second by the next 'slens' and so on. OUTATT does not attempt to write more than 'bufsiz' characters to 'chars', nor does it fail on account of there being more than this to write; if there is too little space, it returns 'ifail' as zero so that it can return the lengths of the strings in 'slens'. As many string fields as fit into less than 'bufsiz' are also returned in 'chars'. Thus the values in 'slens' should be compared with 'bufsiz' after a call to OUTATT; if the sum of the lengths of the string fields up to the last one required is greater than 'bufsiz', then a second call to the function, using more space (how much more being computed from the values in 'slens') in 'chars', will return all the required data.

Can be called from the GO.

OUTBUB - Output rudimentary bulletin board information

Receives

```
KI_cod_logical    *empty  true if bulletin board is to be emptied after
                          being output
```

Returns

```
KI_tag_list_tag   *nwtags tags of new entities
KI_tag_list_tag   *chtags tags of changed entities
KI_tag_list_tag   *dltags tags of deleted entities
KI_tag_list_int   *nwtyps types of new entities
KI_tag_list_int   *chtyps types of changed entities
KI_tag_list_int   *dltyps types of deleted entities
```

```
<KI_tag_list_int> *nwufds user fields of new entities
<KI_tag_list_int> *chufds user fields of changed entities
<KI_tag_list_int> *dlufds user fields of deleted entities
KI_cod_error      *ifail  error code
```

Specific Errors

```
KI_bad_entity_event_comb bulletin board has bad entity/event combination
KI_bulletinb_is_off      bulletin board is not active
```

Description Outputs bulletin board information on entities created, deleted or changed since the bulletin board was last emptied. Depending on the value of 'empty', optionally empties the bulletin board after reading it.

Faces, edges and vertices are bulletined when they are created, deleted and when their topology or geometry is changed (event types BBEVCR, BBEVDE and BBEVCH, respectively). Assemblies and bodies are bulletined only when created and deleted (types BBEVCR and BBEVDE). This routine cannot be used to output the bulletin board if the entity event combinations are not precisely this (see SEBBCO for details). If the event entity combination is invalid KI_bad_entity_event_comb will be returned in ifail.

Three categories of data are returned, relating to created, changed and deleted entities. For each category, three lists are returned giving the tag, type and (optionally) the user field for each entity. No entity will be returned in more than one category, nor will it be repeated within the same list. The order of entities within a list is arbitrary, except that the order is the same for the tag, type and user field lists within a given category.

When an entry is made in the bulletin board, a check is made to see if the same entity is already recorded, and the two records are merged as follows:

	Second Record		
First Record	**Created**	**Changed**	**Deleted**
Created	(1)	created	(2)
Changed	(1)	changed	deleted
Deleted	(1)	(1)	(1)

where:-

(1) this case cannot occur.

(2) the entity will be removed from the bulletin board entirely.

All entries made since the bulletin board was last emptied will be checked in this way.

The type lists contain tokens from the range TYTO00.

User-field data is returned in the three integer lists, such that for entry number "n" in the corresponding tag list, the user field is to be found in entries "(n-1)*ufd_size+1" to "n*ufd_size" inclusive, where "ufd_size" is the user-field size as set by STAMOD.

Whether user-field data is returned depends on the setting of the SLIPBB interface parameter (see SEINTP for details) - when user-field data is not returned, the three integer lists are returned as the NULTAG.

This is the only situation when NULTAG will be returned; in all other cases, lists will be returned, even if they are empty.

OUTCUR - Output curve

Receives

```
KI_tag_curve     *curve     curve to be output
```

Returns

```
KI_cod_tycu      *cutype    type of curve
KI_vec            vec1      first vector defining curve
KI_vec            vec2      second vector defining curve
KI_vec            vec3      third vector defining curve
double           *d1        first double defining curve
double           *d2        second double defining curve
KI_cod_error     *ifail     failure code
```

Specific Errors

```
KI_wrong_sub_type    not a curve supported by this routine
```

Description The type-code of the curve, from the range TYCU00, indicates how to interpret the remaining values as follows:

Type	Argument	Definition
Straight line	(TYCUST) \`vec1'	position on line
	\`vec2'	direction on line
Circle	(TYCUCI) \`vec1'	center
	\`vec2'	axis direction
	\`d1'	radius
Ellipse	(TYCUEL) \`vec1'	center
	\`vec2'	major axis
	\`vec3'	minor axis
	\`d1'	major radius
	\`d2'	minor radius
Intersection	(TYCUIN) \`vec1'	start of curve
	\`vec2'	end of curve

For other curve types only the type-code is returned. Use this code to select the appropriate output routine.

Arguments surplus to an above curve definition are set to zero.

Can be called from the GO.

OUTLEN - Outputs the tolerance value associated with an entity

Receives

```
KI_tag_entity     *entity    can be face, edge or vertex
```

Returns

```
double            *tol      returned tolerance value
KI_cod_error      *ifail    failure indicator
```

Specific Errors

```
KI_wrong_entity            unsuitable entity type
```

Description This function outputs the tolerance value associated with 'entity'. The entity can be any of face, edge or vertex. Refer to SETLEN for the meaning of the tolerance in each case.

Can be called from the GO.

OUTPOI - Output point

Receives

```
KI_tag_point            *point      point to be output
```

Returns

```
KI_cod_typt             *pttype     type of point
KI_vec_position          defn       point coords
KI_cod_error            *ifail      failure code
```

Specific Errors

```
KI_wrong_sub_type            unknown type of point
```

Description The type and coordinates of the point are returned.

The type-code will be from the range TYPT00. At present the only value is TYPTCA (Cartesian point).

Can be called from the GO.

OUTRAN - Output transformation

Receives

```
KI_tag_transform    *transf       transformation to be output
```

Returns

```
double               matrix[16]   transformation matrix array
KI_cod_error        *ifail        failure code
```

Description The current value of the transformation matrix is output in a linear array.

The matrix operates as a post-multiplier on row vectors containing homogenous coordinates thus:

```
(x',y',z',s')  = (x,y,z,s) T
```

where the conventional 3-d coordinates are

```
(x/s,y/s,z/s).
```

The matrix thus consists of

$$\begin{bmatrix} & R & & ,0 \\ & & & ,0 \\ & & & ,0 \\ Tx, & Ty, & Tz, & S \end{bmatrix}$$

R = a rotation/reflection matrix

T = a translation vector

S = a scaling factor

The matrix is filled as follows:

Positions in Array	Contents
0 through 2	rotation/reflection elements r_{11} r_{21} r_{31}
4 through 6	rotation/reflection elements r_{12} r_{22} r_{32}
and 8 through 10	rotation/reflection elements r_{13} r_{23} r_{33}
3, 7 and 11	translation vector
12 through 14	0.0
15	scale

OUTRCU - Outputs a trimmed curve

Receives

```
KI_tag_curve       *trimmed_curve   trimmed curve to be output
```

Returns

```
KI_tag_curve       *basis_curve     underlying curve
KI_vec_position     point_1         start point
KI_vec_position     point_2         end point
KI_dbl_parameter   *parm_1          start parameter
KI_dbl_parameter   *parm_2          end parameter
KI_cod_error       *ifail           failure code
```

Specific Errors

```
KI_bad_type     supplied curve is not a trimmed curve
```

Description Given the tag of a trimmed curve, OUTRCU will return the trimmed curve's data.

A trimmed curve consists of:

- An underlying curve ('basis_curve').
- A pair of points, with their parameters on 'basis_curve', representing the start and end of the trimmed curve.

The following data is output:

- The tag of the basis curve ('basis_curve').
- The start point of the trimmed curve ('point_1').
- The end point of the trimmed curve ('point_2').
- The parameter of the start point of the trimmed curve ('parm_1').
- The parameter of the end point of the trimmed curve ('parm_2').

Note: If 'basis_curve' is closed and the trimmed curve starts and ends at the same point, the parameters output will be a period apart. 'parm_2' will always be greater than 'parm_1'

OUTSFA - Output Trimmed Surface Representation Of Face

Receives

```
KI_tag_face          *face          face to be represented
<KI_int_nitems>      *nopts         number of options in 'iopts'
<KI_cod_srop>         iopts[nopts]  options
<KI_tag_list_dbl>     optdta[nopts] option data lists
```

Returns

```
KI_tag_surface       *surface       surface underlying SP-surves
<KI_int_nitems>      *ntrims        number of non-empty trim loop sets
<KI_tag_list_<list>> *spcus         lists of trimmed SP-curves
<KI_tag_list_<list>> *geoms         lists of corresponding geometries
<KI_tag_list_<list>> *topols        lists of corresponding topologies
KI_cod_error         *ifail         failure indicator
```

Specific Errors

```
KI_cant_make_trimmed_sf  Failed to make up trimmed surface
KI_trim_loop_degenerate  trimming loop degenerate, tolerance may be
                         too big
KI_tolerance_too_tight   tolerance too tight
KI_bad_option_data       Option data incorrect for specified options
KI_cant_make_bspline     Failed to make equivalent bspline surface
KI_missing_geom          Supplied face has no associated surface
```

Description This function will output a trimmed surface representation of a single face.

The Trimmed Surface Representation A trimmed surface represents a bounded region of a parameterised surface by expressing the boundary curves of the trimmed surface as curves in surface parameter space. These curves, referred to as SP-curves, represent a 3-space curve with B-spline surface parameter space curves (i.e. B-spline curves with (u,v) vertices). In cases where the bounded region is the whole of the surface the trimmed surface can optionally be represented as a surface with no trimming curves.

Wire edges and biwire edges are not considered, by OUTSFA, to form part of the boundary of a face and so will not be represented amongst the SP-curves returned.

A description of a single trimmed surface (ntrims = 1) is comprised of the surface to be trimmed, and trimming information held in three lists of lists which are identical in structure to each other, differing only in the type of item that they organise. An application can find three corresponding data by examining the same place in the structure of each of the three lists. These three related pieces of information are: a trimmed SP-curve, the 3-space geometry which it describes, and the topology of the model, if any, on which the 3-space geometry may be found. If the trimmed surface has no trimming loops these lists will be returned as null tags.

spcus:

A list of lists of trimmed SP-curves (that is, each curve is a trimmed curve of type TYCUTR whose underlying curve is an SP-curve of type TYCUSP).

Each list of trimmed SP-curves describes one of the boundary loops of the trimmed surface (these will not necessarily be in 1-1 correspondence with the loops on the face). Where it is appropriate to do so the code will place the outer boundary loop of the trimmed surface at the head of the list of loops. An example of where this is not appropriate is a trimmed surface on a sphere, in this case it is sometimes hard to distinguish between outer boundaries and holes.

Each component trimmed SP-curve of a trimming loop will be G1 in in 3-space. This means, however, that it will not necessarily be possible to represent a single G1 edge curve as a single G1 SP-curve when the surface does not have a parameterisation with continuous first derivatives.

Where an SP-curve is to represent a 3-space curve identified as a line of constant surface parameter the SP-curve description of it will be in the form of a linear B-spline.

geoms:

An optional list of lists of corresponding 3-space geometry. Each 3-space geometry will be either a trimmed curve or a point. Points will arise in cases where a trimmed SP-curve has been produced to cross a degeneracy in the surface parameterisation.

topols:

An optional list of lists of corresponding model topology. Each topology may be one of fin, vertex or null. Vertices will be associated with parameterisation degeneracies. The topology will be null if the trimmed SP-curve represents part of a seam line on the surface underlying the face.

If the trimmed surface description of the face requires that it be split into more than one trimmed surface ('ntrims' > 1), then the three output lists ('spcus', 'geoms', and 'topols') will be embedded one further level. Take for example the case of 'ntrims' = 2. Here each of 'spcus', 'geoms', and 'topols' will be a list of two lists, and those sub-lists will refer to separate trimmed surfaces and will each have the structure described for the output in the case of 'ntrims' = 1. A complete description of the first trimmed surface can be found by taking the first entry in each of 'spcus', 'geoms', and 'topols'. Taking the second entry from 'spcus', 'geoms', and 'topols' will similarly give the description of the second trimmed surface.

Note: Iin cases where 'ntrims' = 0 there is actually one trimmed surface namely the whole of the underlying surface given in the returned surface. In such cases the trimming loop is absent because it doesn't actually trim out any of the surface and the options selected permitted its omission.

The surface that is trimmed is returned in 'surface' and the sense of the surface is set so that the surface normal points in the same direction as the normal on the original face ('face').

Deleting Returned Geometry In the returned data from OUTSFA all geometries are either copies of geometry on the face or have been built specifically as alternative representations of that geometry. To avoid the modeller filling up with unrequired nodes the geometry returned in 'surface', 'spcus' and 'geoms' should be deleted as soon as its usefulness expires.

Options OUTSFA supplies various surface and SP-curve options, which are selected by option tokens received in 'iopts'. For some options, data is also required. This data is supplied by a real list whose tag is passed in 'optdta', in an array position corresponding to the 'iopts' entry. If no data is required for an option a null tag should be passed.

Surface Related Options: The Surface whose parameter space is to be used can either be the surface attached to the face, or a B-spline representation of that surface. The options, and option tokens and data are as follows:

Option	Option Token	Option Data
To use Surface attached to face/compos ite (default option)	no token	none
To use a B-spline approximation	SROPBS	tolerance
Prohibit extension of the surface to fit SP-curves that stray outside	SROPNE	None

where the tolerance is the maximum distance between the B-spline and the surface it represents.

For the B-spline option there are two secondary options available. These options can only be supplied in addition to SROPBS. The options are:

Option	Token
Force output of B-spline of degree 3	SROPCU
Force output of non-rational B-splines	SROPNR

The effect of these options on the B-spline surface representations produced for different surface types is as documented in the OUBSFA documentation.

No matter which type of surface is requested for output, OUTSFA will by default extend it to fit SP-curves approximating the face boundaries that stray outside of the natural boundaries of the surface. The token SROPNE is provided to switch off this default behaviour. When SROPNE is set OUTSFA will succeed as before, however the returned surface will not be extended to include all SP-curves in the output that pass outside the original bounds of the surface's parameter space. Since SROPNE may lead to output where SP-curves that pass outside the natural boundaries of the surface on which they lie, any application using this token must be able to deal with them.

SP-curve Related Options There are 3 options available for the SP-curves which are to be constructed. For all options, a default action is taken if an option token and data is not supplied. Where more than one token exists for an option they are mutally exclusive, i.e. only one may appear as a parameter to OUTSFA. The option tokens and related option data (which must be supplied with the option) are as follows:

Option	Token	Data	Description
Tolerance	SROPCT	1 real tolerance' c.f. below	Curve Tolerance- - Allows specification of the tolerance which the SP-curve representations of the edges should satisfy. Default is 10,000*modeler resolution.
Configuration	SROPCN	none	Confined, No- - Trimming loops will not be confined to a single period on periodic faces. The trimmed surface returned may have more than one outer boundary (e.g. ends of a cylinder). The trimming loops will be continuous in parameter space, but not be closed.
	SROPCY	none	Confined, Yes- - Trimming loops will be confined to a single period on periodic faces. The trimmed surface returned may have more than one outer boundary (e.g. the ends of a cylinder). The trimmed SP-curves in a loop may have gaps between them in parameter space where there are degeneracies.
	SROPCC	none	Confined and Closed- - Trimming loops will be confined to a single period on periodic faces. All the trimming loops will be closed and without gaps in parameter space. Each trimmed surface will have no more than one outer peripheral loop. The outer peripheral loop may be omitted if it does not trim off any of the surface. This is the default action.
	SROPCP	none	Confined with Periphery- - Trimming loops will be confined to a single period on periodic faces. All the trimming loops will be closed and without gaps in parameter space. Each trimmed surface will have exactly one outer peripheral loop .
Degeneracies	SROPED	none	Exclude Degeneracies- - Don't represent degeneracies except where the selected configuration option applies this. This is the default action.
	SROPID	none	Include Degeneracies- - All parametric degeneracies occurring on the face will be represented (irrespective of whether they are associated with topology).

As indicated above the SP-curve tolerance may be specified using the SROPCT option. The tolerance given with SROPCT is a distance tolerance in model units: this refers to the maximum allowable distance between an SP-curve and the edge curve that it represents. Although extensive tolerence checking is carried out, and the accuracy of the representation will usually satisfy the supplied tolerance, this cannot be guaranteed.

When using a B-spline surface approximation, care must be taken when supplying the tolerance. The surface approximation needs to be accurate enough for the SP-curves to satisfy their tolerance. It is suggested that the surface B-spline tolerance should be no greater than half the distance tolerance supplied for the SP-curves.

The tolerance is ignored for edges of the face that are already tolerant and have an SP-curve of degree 1 or 2 (lying in the surface to be returned) attached. In this case the appropriate trimmed section of the present SP-curve will be returned without approximation.

If the required tolerance cannot be met then the function will fail with error KI_tolerance_too_tight indicating that a larger tolerance may allow a successful approximation. Similarly there are occasions on which the curve tolerance specified may be so large that the face will appear, to OUTSFA, to be nothing more than a wire. In such cases, rather than output degenerate loops, OUTSFA will return the ifail KI_trim_loop_degenerate. Setting a smaller tolerance may result in a successful attempt to produce trimmed surface output.

Output Related Options

There are just two options related to the quantity of data returned. Each option is provided to indicate that one of the two output lists holding information associated with a trimming loop is required.

Token	Description
SROPNG	Need Geometry - - Requests the return of the list of geometries ('geoms') associated with the SP-curves in the trimming loops. (The list is by default not returned)
SROPNT	Need Topology - - Requests the return of the list of model topologies ('topols') associated with the SP-curves in the trimming loops. (The list is by default not returned)

OUTSUR - Output surface

Receives

```
KI_tag_surface    *surfac   surface to be output
```

Returns

```
KI_cod_tysu       *sftype   surface type
KI_vec             vec1     first vector defining surface
KI_vec             vec2     second vector defining surface
double            *d1       first double defining surface
double            *d2       second double defining surface
KI_cod_logical    *sense    surface sense. see below
KI_cod_error      *ifail    failure code
```

Specific Errors

```
KI_wrong_sub_type   not a surface supported by this routine
```

Description The 'sense' and 'sftype' arguments are returned for all surface types. Further information is returned according to the value of 'sftype':

Type	Arg	Definition	normal if sense == KI_true
Plane	(TYSUPL) `vec1	position in plane	same as `vec2'
	`vec2'	normal direction	
Cylinder	(TYSUCY) `vec1'	position on axis	away from axis
	`vec2'	axis direction	
	`d1'	radius	
Cone	(TYSUCO) `vec1'	position on axis	away from axis
	`vec2'	axis direction	
	`d1'	radius of cone at `vec1'	
	`d2'	half angle (in radians)	
Sphere	(TYSUSP) `vec1'	center position	away from center
	`d1'	radius	
Torus	(TYSUTO) `vec1'	center position	away from circle described by major axis
	`vec2'	axis direction	
	`d1'	major radius	
	`d2'	minor radius	
Blend	(TYSUBL) 'd1'		

For all other surface types only the type-code is returned. Use this code to select the appropriate output routine.

If sense == KI_false then the surface normal is opposite to the normal described for sense == KI_true.

Where no information is returned, the relevant argument is set to zero.

Can be called from the GO.

OUUFEN - Return user field of entity

Receives

```
KI_tag        *tag      tag whose user field is required
```

Returns

```
KI_int_ufdval  ufdval[]  user-field value - array length is the
                         user-field size set by STAMOD
KI_cod_error  *ifail     error code
```

Specific Errors

```
KI_not_a_tag             'tag' is not valid
KI_no_user_fields        user fields are not in use
```

Description The user-field value for 'tag' is returned in array 'ufdval'.

If the user field has not been set explicitly by calling SEUFEN, each element of array 'ufdval' will be zero.

Can be called from GO.

PICKEN - Pick entities inside a cylindrical volume

Receives

KI_tag_list_part	*palist	part or list of parts
<KI_tag_list_<transform>>	*transf	transformation or list of transformations
KI_vec	point	point through which axis asses
KI_vec_direction	axis	axis of cylindrical volume
KI_dbl_radius	*rad	radius of cylindrical volume
KI_cod_slpk	*opt	ray options
KI_cod_ty	*entype	type of entity to pick

Returns

<KI_int_nitems>	*nhiten	number of picked entities
<KI_tag_list_entity>	*hitlis	list of picked entities
<KI_tag_list_int>	*indlis	list of indices of the owning parts
<KI_tag_list_dbl>	*dislis	list of minimum distances from picked entities to ray
<KI_tag_list_dbl>	*hitpts	points on ray axis
KI_cod_error	*ifail	failure indicator

Specific Errors

KI_bad_type	check ray type is correct; entity in part list is of incorrect type
KI_wrong_transf	transform contains scaling or reflection
KI_wrong_entity_in_list	entity in part list is of incorrect type
KI_not_same_length	part and transform lists are of different engths

Description PICKEN provides assistance in selecting entities from a display. PICKEN fires a thick ray at a list of parts and can return any edge, vertex, construction curve or construction point hit. The inverse of a transform in 'transf' is applied to the ray before it is fired at the corresponding part in 'entys'; i.e. PICKEN acts as if each part in 'entys' is transformed (by the corresponding transform in 'transf') before firing the ray, without actually transforming them.

If none of the parts are to be transformed, 'transf' may be specified as a null tag. A null tag in 'transf' is interpreted as the identity transformation.

The volume from which the entities will be picked is defined as a cylinder of infinite length. The axis of the cylinder passes through 'point' and runs parallel to 'axis'. The radius of the cylinder is 'rad'.

'entype' is the type code which determines what type of entity to look for. If a body or list of bodies is given, vertices, edges, construction points and construction curves can be found, so either TYTOVX, TYTOED, TYGEPT or TYGECU can be given. If an assembly or list of assemblies is given, only construction points and construction curves can be found, so either TYGEPT or TYGECU can be given.

'opts' is the type code which determines whether the ray is to be infinite SLPKIR or semi_infinite SLPKSR. If SLPKIR is selected then PICKEN will return all entities along the whole ray. If SLPKSR is selected then PICKEN will return entities along a ray starting at 'point' and in the direction 'axis'.

All the entities of the correct type which lie within the defined region will be found. The data will be returned in three lists of length 'nhiten' as follows:

'hitlis' - A list of tags of picked entities

'indlis' - A list of indices of the owning bodies or assemblies

'dislis' - A list of minimum distances from picked entity to the defined axis

'hitpts' - A list of points on the axis corresponding to the minimum distances in 'dislis'. These points are the projections of the minimum distance points on to the axis.

The entries in the lists will be sorted according to the values in the 'dislis' list, so that the nearest entity comes first.

If no suitable entities are found, the value of 'nhiten' will be zero and 'hitlis', 'indlis', 'dislis' and 'hitpts' will be returned as the null tag.

The distances are not determined to modelling accuracy, and so this routine should only be used for operations such as picking from a display, and not for modelling calculations.

If transforms are supplied they may only contain translation and rotation components. Reflections, scales and shears are not allowed.

PIERCE - Remove face from sheet

Receives

`KI_tag_face`	`*face`	`face to remove`

Returns

`KI_cod_error`	`*ifail`	`failure indicator`

Specific Errors

`KI_bad_wire`	`invalid wire would result`
`KI_fragment`	`sheet would break apart`
`KI_not_sheet`	`body is not a sheet`

Description This routine deletes a face from a sheet body. The given face must be a face of a sheet body, with or without a surface attached. Removal of the face must not break the sheet into two or more parts.

If the body has more than one face, it remains a sheet body; edges and vertices belonging to the face, but which are shared with no other face, will be deleted along with it.

If the face is the only face of the sheet body, the result will be either a wire or minimal body; none of the body's edges and vertices will be deleted by the operation. The edges and vertices must form a valid wire or minimal body, i.e. they must be connected (in a single loop or a pair of bi-wire loops) and no vertex may be used by more than two edges.

PTENFE - Put entities into feature

Receives

```
KI_tag_feature           *featre    feature
KI_tag_list_entity       *entity    entities to be added
```

Returns

```
KI_cod_error             *ifail     failure code
```

Specific Errors

```
KI_duplicate_list_item   entity appears twice in argument list
KI_already_in_feat       entity is already in feature
KI_not_in_same_part      entity and feature not in same part
KI_wrong_type_for_feat   wrong entity in list
```

Description 'entity' may be either a single entity, or a list of entities. All the entities are added to the feature, provided they are of a permitted type, are contained in the same part as the feature and are not in the feature already. If any entity does not satisfy these criteria, an ifail will be returned, and the feature will be unchanged.

If the feature belongs to an assembly the permitted types for 'entity' are instance, surface, curve and point.

If the feature belongs to a body the permitted types for 'entity' are region, face, edge, vertex, surface, curve and point.

Unless the feature is mixed (i.e. feature type is TYFEMX) its type must correspond to that of 'entity' (TYFESU for a surface etc.).

PTINLI - Put values into a list of integers

Receives

```
KI_tag_list_int *list         list in which to put items
<KI_int_index>  *startx       position in list where first value is
                              to be put
KI_int_nitems   *nvals        number of values to be put into list
int              ivals[nvals] values to put into list
```

Returns

```
KI_cod_error          *ifail      failure indicator
```

Specific Errors

```
KI_bad_index   'startx' is not in range 1 to list length + 1
```

Description The given values are put into the list. The first value goes into the 'startx' position, the second into 'startx' + 1 and so on. The first element in the list is number 1.

'startx' must not be greater then the current length of the list plus 1.

If 'startx' is zero then the given values are appended to the end of the list.

If 'startx' is in the range[1, length of list], existing elements will be overwritten.

Can be called from the GO.

PTRLLI - Put values into a list of reals

Receives

KI_tag_list_dbl	*list	list in which to put items
<KI_int_index>	*startx	position in list where first value is to be put
KI_int_nitems	*nvals	number of values to be put into list
double	rvals[nvals]	values to put into list

Returns

KI_cod_error	*ifail	failure indicator

Specific Errors

KI_bad_index	'startx' not in range 1 to list length + 1

Description The given values are put into the list. The first value goes into the 'startx' position, the second into 'startx' + 1 and so on. The first element in the list is number 1.

'startx' must not be greater then the current length of the list plus 1.

If 'startx' is zero then the given values are appended to the end of the list.

If 'startx' is in the range[1, length of list], existing elements will be overwritten.

Can be called from the GO.

PTTGLI - Put values into a list of tags

Receives

KI_tag_list_<tag>	*list	list in which to put tags
<KI_int_index>	*startx	position in list where first value is to be put
KI_int_nitems	*ntags	number of tags to be put into list
KI_tag	tags[ntags]	tags to put into list

Returns

KI_cod_error	*ifail	failure indicator

Specific Errors

KI_bad_index	'startx' not in range 1 to list length + 1

Description The given values are put into the list. The first value goes into the 'startx' position, the second into 'startx' + 1 and so on. The first element in the list is number 1.

'startx' must not be greater then the current length of the list plus 1.

If 'startx' is zero then the given values are appended to the end of the list.

If 'startx' is in the range[1, length of list], existing elements will be overwritten.

PTTGLI does not perform any check on the validity of the tags to be inserted.

Can be called from the GO.

RAYFIR - Intersect ray with bodies

Receives

```
KI_tag_list_part          *palist entities to fire ray through
<KI_tag_list_<transform>> *transf part transformations
<KI_int_nitems>           *wchhit number of intersections wanted
KI_vec                     point  point from which ray is fired
KI_vec_direction           direct direction of ray
```

Returns

```
<KI_int_nitems>           *nhitpt number of points returned
KI_tag_list_dbl           *hitpts points where ray hit bodies
KI_tag_list_face          *hitfas faces hit by ray
KI_tag_list_int           *indces indices of bodies hit by ray
KI_cod_error              *ifail  failure indicator
```

Specific Errors

```
KI_wrong_transf          Unsuitbale transform
KI_not_same_length       Entity and transform lists are of different
                         lengths
KI_wrong_entity_in_list  Entity is not a part
```

Description RAYFIR fires a ray at a list of transformed parts and returns the faces hit.

A face is considered to be hit by the ray if and only if the ray passes through the face. Cases where the ray touches the face but does not pass through it are not considered hits, e.g. a tangential intersection of the ray with a face is not a hit.

This is achieved internally by first applying the inverse transform to the ray, and then applying the original transform to the hit points found. If none of the parts are to be transformed, 'transf' may be specified as a null tag, which will be interpreted as an identity transform.

The ray begins at 'point' and proceeds along 'direction' as far as necessary: faces in the opposite direction will not be found.

For each intersection found, the routine will insert in the lists it returns:

- The coordinates of the point of intersection, in the next three entries in 'hitpts'.
- The tag of the face intersected by the array, in the next entry of 'hitfas'.
- The index in 'palist' of the particular part owning the face hit by the ray, in the next entry of 'indces'.

Thus the lengths of 'hitpts', 'hitfas' and 'indces' will be 3*'nhitpt', 'nhitpt' and 'nhitpt' respectively.

If the ray does not intersect any of the transformed parts three empty lists will be returned.

If non-empty lists are returned then they will be sorted according to ascending distances measured from `point' along the sense of `direct'. If the ray hits two faces at the same point, for instance at an edge, then the relative ordering of the pair is arbitary.

The number of intersections returned is controlled by the parameter 'wchhit'. At present the values are:

0 - All intersections with all bodies

n > 0 - The closest n intersections with the ray to the point from which the ray is fired along the sense of 'direct'.

If transforms are supplied they may only contain translation and rotation components. Reflections, scales and shears are not allowed.

REDINS - Redirect instance

Receives

```
KI_tag_instance       *instnc    instance to redirect
KI_tag_part           *part      part it should instance
```

Returns

```
KI_cod_error          *ifail     failure indicator
```

Specific Errors

```
KI_anon_sub_part           instance of anonymous sub-part of stored part
KI_cyclic_assy             instance would cause cyclic reference
KI_not_in_same_partition   instance and part are in different patitions
```

Description REDINS redirects 'instnc' so that it instances a different part. It is equivalent to creating a new instance of 'part', with the same transform (i.e. with the same tag) and attributes attached, and then deleting the 'instnc'.

If the redirected instance would cause the assembly graph to become cyclic (i.e. when redirecting an instance in A to P, and P or a sub-part of P is A) KI_cyclic_assy will be returned in 'ifail'.

We define the true-sub-parts of a stored (ENSTST) part S as those anonymous (ENSTAN) sub-parts of S reachable from S without encountering other stored parts.

If P is anonymous and is a true-sub-part of some stored part S then P may only be instanced from S or a true-sub-part of S. If this condition is not met KI_anon_sub_part is returned in 'ifail'.

REEDSH - Replaces the edges of a sheet body

Receives

```
KI_tag_body           *sheet           Sheet body
KI_dbl                urange[2]        urange of face
KI_dbl                vrange[2]        vrange of face
```

Returns

```
KI_cod_error          *ifail           failure code
```

Specific Errors

```
KI_wrong_surface      surface of sheet is of wrong type
KI_bad_parameter      invalid parameter range
KI_unsuitable_entity  sheet has more than one non-rubber face
KI_missing_geom       sheet has no surface attached
KI_not_sheet          body is not a sheet
```

Description The supplied body, `sheet' must be a sheet body which must only have one face and there must be a surface attached to the face.

The edges of the sheet are removed and replaced by edges corresponding the isoparameter lines defined by 'urange' and 'vrange' (ie the new face is parametrically rectangular).

For each pair of parameter ranges, the following rules apply:

- The first element must be less than the second.
- Both elements must lie inside the parameter range of the surface of the sheet, as given by ENCUPA/ENSUPA, unless the corresponding parameter is periodic. In that case the first must lie in the range, and the difference between the two may not exceed the period. The resulting face will straddle the boundary of the parametrisation.
- fr
-

> **Note:** REEDSH will not replace the edges of a blend face (type TYSUBL).

RESUSH - Replaces the surface of a sheet body

Receives

KI_tag_body	*sheet	Sheet body
KI_tag_surface	*surf	new surface for sheet body
KI_dbl_distance	*tol	tolerance of SP_curves

Returns

<KI_tag_list_edge>	*edges	edges converted to SP_curves
<KI_int_nitems>	*nedges	number of converted edges
KI_cod_error	*ifail	failure code

Specific Errors

KI_invalid_geometry	surface fails continuity checks
KI_bad_tolerance	edge too short for tolerance
KI_failed_to_replace	unable to replace surface of sheet
KI_bad_sharing	illegal sharing of surface
KI_wrong_surface	surface is incompatible
KI_unsuitable_entity	sheet has more than one non-rubber face
KI_missing_geom	sheet has no surface attached
KI_not_sheet	body is not a sheet
KI_not_in_same_partition	sheet and surf are in different partitions

Description The surface of the sheet is tweaked to the new surface while the parameter space representation of the edges is maintained.

The supplied body, 'sheet' must be a sheet body which must have only one face and there must be a surface attached to the face.

The new surface, 'surf' must be of the same type as the surface currently attached to the face of the sheet or an offset of that type. The new surface must also have the same bound classification, as given by ENSUPA, as the surface of the face and must be capable of passing the checks imposed by CHCKEN. Continuity checks are also performed (see ENDIPE) if the appropriate option is set (see SEINTP).

If the curve of an edge is an SP-curve (TYCUSP) or a constant parameter curve of the surface of the face then it will be transfered to the new surface.

All other edges will be converted to SP-curves using the supplied tolerance, 'tol' and then transfered to the new surface. It is an error if any of these edges are shorter than twice the tolerance. During this conversion the edges may need to be split at surface discontinuities.

All the edges whose curves have been converted to SP-curves will be returned in 'edges'.

> **Note:** RESUSH will not replace the surface of a blend face (type TYSUBL).

RETLEN - Restores Parasolid tolerance to the supplied edge

Receives

```
KI_tag_entity         *entity      edge
```

Returns

```
KI_cod_rttl           *retcod      status of operation
KI_cod_error          *ifail       failure indicator
```

Specific Errors

```
KI_wrong_entity            unsuitable entity type
```

Description RETLEN attempts to remove the tolerance value associated with the given edge, restoring Parasolid default tolerance and replacing fin curves by a single 3-space curve attached to the edge. This operation may not always be possible - the level of success is indicated by the status code 'retcod' which can take the following values:

retcod value	status of operation
RTTLOK	successfully replaced tolerance and geometry
RTTLNT	neither tolerance nor geometry replaced as surfaces are approximately tangent at edge
RTTLMG	not enough geometry present on edge or adjacent faces to recompute edge geometry
RTTLRF	recomputation of edge geometry failed

RETLEN attempts to replace geometry and tolerance as follows:

- if all adjacent faces of the edge have surface geometry attached, a 3-space curve of intersection will be derived by re-intersection of these surfaces. Extensive pre-checks will be made to ensure that tangent or near-tangent configurations (such as boundaries of Variable Radius Blends) are not attempted in this way. In these cases the token RTTLNT will be returned, and the edge geometry and tolerance will be unchanged. There will also be cases where the edge is not a near tangency, but nevertheless a new curve of intersection could not be computed (for example, the adjacent surfaces may simply not intersect to Parasolid default resolution); in such cases RTTLRF will be returned. Otherwise, if RETLEN is successful, 'retcod' will be set to RTTLOK.

The particular simple case where one of the SP-curves lies within Parasolid default tolerance of the adjacent surface will be dealt with by either simplifying this SP-curve to its 3-space equivalent, or failing this, by transferring the SP-curve to the edge.

- if not all face geometries are present, as at the boundary of a sheet body, the SP-curve present on the relevant fin will be first simplified to its 3-space equivalent (if it was a constant parameter curve) and this new curve attached to the edge. If no simplification was possible, the SP-curve itself will be transferred to the edge. In either case RTTLOK will be returned in 'retcod' if the geometry was successfully recomputed.

If insufficient geometry was present to successfully replace the tolerance of the edge, RTTLMG (missing geometry) will be returned.

RETLEN will also attempt to set the tolerance of each vertex of the edge to Parasolid default. It will only be able to do this if all the other edges meeting at the vertex in question have Parasolid default tolerance associated with them, and the new 3-space curve intersects at a unique point with all the other edge curves meeting there.

> **Note:** If both vertex tolerances are reset, the curve attached to the edge will no longer be a trimmed curve (type TYCUTR).

RMFASO - Remove faces into new solids

Receives

```
KI_tag_list_face   *faces   faces to be removed
KI_cod_sllo        *actpar  action to mend holes on parents
                             SLLOCP => cap
                             SLLOGR => grow
                             SLLORB => leave rubber
KI_cod_sllo        *actoff  action to mend holes on offspring
                             SLLOCP => cap
                             SLLOGR => grow
                             SLLOGP => grow from parent
                             SLLORB => leave rubber
```

Returns

```
KI_tag_list_body   *parnts  parent body fragments
KI_int_nitems      *nprnts  number of parent fragments
KI_tag_list_int    *sprnts  state of parent fragments
                             RTLOOK => Valid
                             RTLONG => Negated
                             RTLOSX => Self-Intersecting
KI_tag_list_body   *offspg  offspring bodies
KI_int_nitems      *nofspg  number of offspring
KI_tag_list_int    *sofspg  state of offspring
                             RTLOOK => Valid
                             RTLONG => Negated
                             RTLOSX => Self-Intersecting
KI_cod_error       *ifail   failure code
```

Specific Errors

KI_wire_body	Unable to make solid from wire body
KI_all_faces_in_body	Cannot remove all faces from body
KI_dont_make_solid	Unable to make solid from faces
KI_cant_heal_wound	Can't heal wound - impossible geometry
KI_non_manifold	Cannot heal wound with non-manifold boundary
KI_not_in_same_shell	Faces not all from the same shell
KI_general_body	General body unsuitable for sweep

Description The 'faces', which must all belong to the same body, are separated from it, and used to make one or more new bodies. The new bodies formed from 'faces' are termed "offspring", and the original body, which may become fragmented, "parent fragments".

The faces must not form the shell of a sheet body. If they form an inner shell (i.e. void) of a solid, that shell is removed into a solid. If the faces do not form a shell, holes left in the parent fragments by the removal of 'faces' are "healed" according to the action code 'actpar'. All holes are healed with the same type of action. Similarly, 'faces' are formed into solid offspring by healing holes in them according to the action 'actoff'. The action "cap" causes a new face to be created for a hole with a surface which fits the edges of the hole. The action "grow" causes the faces around the hole to be extended until the hole is covered. The action "grow from parent" has the same effect as "grow" in parent fragments, however in offspring it causes a hole to be covered with an inverse of the "patch" made by extending faces around the corresponding hole in the parent. It is not necessary for 'actpar' to be "grow" when 'actoff' is "grow from parent". The action "leave rubber" covers each hole with a rubber face.

Restrictions on growing:

- Edges of faces adjacent to a wound which do not form part of the loop of edges around the wound, but have a vertex on it, will not be allowed to contract back from that point. The "shrinkage" option implemented in DELFAS does not work for this routine.
- Each closed loop around a face or group of faces is healed independently. If the only solution would require more than one loop to be healed together it will not be found.
- The wound left by removing all the faces in a shell or body cannot be healed.

In general this procedure may give rise to self-intersecting object boundaries which could cause unpredictable errors later.

If local checking is on (see SEINTP and OUINTP), and the action is not leave rubber, consistency checks will be made on newly created topological and geometrical entities, and the state of the body returned. A state of RTLOOK indicates the body is valid. A state of RTLONG indicates that the result body was originally "inside out" but has been negated, and is now "positive" (has positive volume) and valid. A state of RTLOSX indicates the body is self-intersecting and further modelling operations on it may fail. It is the responsibility of the calling routine to make any necessary reparation.

If the session parameter for local checking is switched off or the action is leave rubber, the state returned will be RTLOOK regardless.

A self-intersecting body can be returned even if the ifail is zero.

This function is not supported for faces on general bodies.

ROLBFN - Rolls back or forward by N steps between roll-marks

Receives

```
int            *nsteps number of steps to be rolled forward (> 0)
                       or back (< 0)
```

Returns

```
int            *asteps actual number of steps rolled forward (> 0)
                       or back (< 0)
KI_cod_error   *ifail  error code
```

Specific Errors

```
KI_not_at_rollmark    state of modeller changed since last roll-mark
KI_roll_forward_fail  roll forward not possible
KI_bad_value          'nsteps' is zero
KI_roll_is_off        logging for roll-back is disabled
```

Description Rolls the model forward or back by 'nsteps'; a positive number indicates roll-forward, a negative number specifies roll-back. Error code KI_bad_value will be returned if 'nsteps' is zero.

Rolling forward or back has the effect of restoring tag memory, modelling parameters, interface parameters and the bulletin board to the state they were in immediately after the call to ROLSMK which set the relevant roll-mark.

The ROLBFN call will affect validity of tags of entities which were created or deleted between the two roll-marks representing the states before and after the ROLBFN call. The modeler will always trap invalid tags, giving error KI_not_a_tag. Note, however, that entities restored by a get-snapshot operation may have the same tags as entities which existed prior to the GETSNP call.

It may be that modeler is unable to roll as many steps as have been requested - this may be because not enough marks have yet been set, or because the logging file is not large enough to record that many steps, or because an attempt is being made to roll forward to a mark which has been invalidated by a call to ROLSMK since the roll-back. Whatever the reason, the modeler will roll back or forward as many steps as it can, and will return the number of steps in 'asteps'. The modeler will not raise an error, even when no steps can be rolled (in this case 'asteps' will be returned as zero).

If the state of the modeler has been changed since the most recent call to ROLSMK, ROLBFN or ROLBLM, ROLBFN will return the error code KI_not_at_rollmark. In this the case, you must either call ROLSMK (to set a new roll-mark) or ROLBLM (to roll back to the previous one) before calling ROLBFN.

Only 1024 rollmarks are stored. It is not possible to roll backwards or forwards further than the 1024 marks set.

Routine OUSTAT can be used to get information about the roll-back system such as how many roll-back and roll-forward steps are available and whether the modeler is at a roll-mark.

ROLBLM - Rolls back changes since the last roll-mark

Returns

```
KI_cod_error    *ifail                          error code
```

Specific Errors

KI_no_rollmark	no previous roll-mark exists
KI_roll_is_off	logging for roll-back is disabled

Description Restores tag memory, modelling parameters, interface parameters, and the bulletin board to the state they were in immediately after the most recent call to ROLSMK or ROLBFN.

Note that the ROLBLM call may affect validity of tags; tags referring to entities created since the last roll-mark will cease to be valid, whilst those referring to entities deleted will become valid. The modeler will always trap invalid tags, giving error KI_not_a_tag. Note, however, that entities restored by a get-snapshot operation may have the same tags as entities which existed prior to the GETSNP call.

ROLBLM cannot be used to roll back from one roll-mark to the previous one; this can only be done by use of ROLBFN. If ROLBLM is called repeatedly, only the first call will have any affect.

It is not possible to roll forward to the state of the model immediately prior to the ROLBLM call; if you want the option of rolling forward, you should instead call ROLSMK to set a mark, and then ROLBFN to roll back to the previous mark.

If the call to ROLBLM follows a roll-back by use of ROLBFN, you will then have the option of doing a roll-forward to undo the effect of the earlier ROLBFN call. This feature allows you to test two alternative modelling operations as follows:

- Set a roll-mark, then try the first operation.
- Set a roll-mark, roll back using ROLBFN, then try the second operation.

Decide which operation to keep.

If you want the second operation, simply set a further roll-mark which will invalidate the roll-mark representing the first operation.

If you want the first operation, call ROLBLM to undo the second operation, then call ROLBFN to roll forward to the result of the first operation.

ROLSMK - Sets a roll-back mark

Returns

KI_cod_error	*ifail	error code

Specific Errors

KI_rollmark_failed	failed to set rollmark
KI_roll_is_off	logging for roll-back is disabled

Description Records the state of the model so that a later roll-back (using ROLBLM or ROLBFN) may return to it.

This mark will last until one of the following happens :

- Several later marks are set. The exact number will depend on the size of the roll-back file, as set in the call to SEINTP. Eventually this mark will not be recoverable, and a call to ROLBFN will return a smaller number of steps than requested.
- ROLBFN is called with a sufficient number of steps to roll back further than this mark, then another mark is set by again calling ROLSMK; a call to ROLSMK invalidates any rollmarks which might otherwise be used for a roll-forward.

RRFCET - Generate facetted rendering

Receives

```
<KI_int_nitems>            *nopts          number of options in 'iopts'
KI_cod_rrop                 iopts[nopts]   rendering options
<KI_tag_list_dbl>           optdta[nopts]  option data lists
KI_tag_list_entity         *entys          entities to render
<KI_tag_list_<transform>>  *transf         entity transforms
```

Returns

```
KI_cod_error               *ifail          failure indicator
```

Specific Errors

```
KI_abort_from_go         Rendering aborted by GO
KI_general_body          General body supplied in list
KI_wrong_transf          Unsuitable transform
KI_bad_view_mx           Option data contains invalid view matrix
KI_not_same_length       Entity and transform lists are of different
                         lengths
KI_wrong_entity_in_list  Entity is not a part or a face
KI_bad_option_data       Option data incorrect for specified options
```

Description RRFCET outputs, through the Graphical Output interface (GO), data for a facetted representation of the entities.

RRFCET can render assemblies, bodies and faces; it cannot render individual edges. General bodies and faces from general bodies are not supported.

Transforms may only contain translation and rotation components. Reflections, scales and shears are not allowed.

The data output is selected by options in 'iopts'. For some options the output is further controlled by data in a real list whose tag is passed in the corresponding entry in 'optdta'. If no data is required for an option a null tag should be passed. For those options which require data, a zero value indicates that the corresponding tolerance or limit is ignored; at least one value must be non-zero.

Options accepted are:-

Option	Contents of option data
RROPTR: TRansform	none
RROPNF: No Fitting	none
RROPVM: Vertex Matching	none
RROPCV: ConVexity	none
RROPHO: HOles permitted	none
RROPVN: Vertex Normals	none
RROPPI: Parameter data	none
RROPD1: Derivative data	none
RROPD2: Derivative data	none
RROPET: Edge Tags	none

RROPST: Surface Tolerance	(1) Distance tolerance in model units (2) Angular tolerance in radians
RROPCT: Curve Tolerance	(1) Chord tolerance in model units (2) Maximum chord length in model units (3) Angular tolerance in radians
RROPFS: Facet Size	(1) Maximum number of sides per facet (2) Maximum width of facet in model units
RROPMF: Minimum Facet size	(1) Minimum size of facet
RROPPT: Planarity Tolerance	(1) Distance tolerance in model units (2) Angular tolerance in radians
RROPFP: Facet Perspective	(1-16) Viewing transformation matrix
RROPFI: Facet Infinite	(1-16) Viewing transformation matrix
RROPTS: Facet Strips	(1) Maximum length of facet strip
RROPIL: Ignore Loops	(1) Number, n, of loops following (2-(n+1) Tags of loops to be ignored
RROPSD: Silhouette Density	(1-3) Viewing direction vector (4-5) Density control

Option RROPTR is required if any of the entities is to be rendered in a transformed position (e.g. as part of an assembly). If it is not specified, the contents of 'transf' are ignored, and entities are rendered in their local coordinate system.

Option RROPNF will not attempt to fit the facets together at the edges of each face. With this option the facetting algorithm will be faster as no clipping and matching will be performed at face edges. This may produce an inconsistent result with overlapping facets or gaps.

Option RROPVM ensures that there are no gaps along model edges and that along these edges there are no facet vertices which are interior to an adjacent facet edge. This option only affects whole bodies and assemblies.

Option RROPCV causes the shape of the facet to be limited so that the sides of the facet form a convex polygon. Each interior angle of a convex polygon is less than pi radians. A convex polygon contains no holes.

Option RROPHO will allow the facet to be represented with holes in its interior.

Option RROPVN should be specified if surface normals at the facet vertices are required.

Option RROPPI should be specified if parameter information at the facet vertices is required.

Option RROPD1 should be specified if first surface derivatives at the facet vertices are required. RROPD2 should be specified if first and second surface derivatives at the facet vertices are required. If either of RROPD1 or RROPD2 is enabled then vertex normals and parameter data will also be output.

If option RROPET is specified, RRFCET will output an array of edge tags, one edge tag for each facet edge, indicating the model edges from which the facet edges are derived. If a facet edge was not derived from the model a null tag is given.

Option RROPST controls the facetted representation of the entity by considering the surface approximation. The first value in the option data is a distance tolerance on a facet. This is an upper bound on the distance from a position on a facet to the surface. The second value is an angular tolerance on a facet. This is an upper bound on the angular deviation between the surface normals at any two positions under the facet. This function will usually satisfy these tolerances although this is not guaranteed. If this option is not specified a default angular tolerance is used.

Option RROPCT controls the facetted representation of the entity by considering the curved edge approximation. A curved edge is approximated by a number of straight lines called chords. The first value in the option data is a chord tolerance. This is an upper bound on the distance from each chord to the curve it is approximating. The second value is a chord limit. This is an upper bound on the length of a chord used in the approximation of a curved edge. The third value is an angular tolerance. This limits the angular error between a curve and a chord used in its approximation. This is an upper bound on the sum of the two angles formed between the chord and the curve tangent at each chord end. If this option is not specified a default angular tolerance is used.

Option RROPFS controls the facetted representation of the entity by considering the size of the facet. The first value in the option data specifies the maximum number of sides in a facet. The second value specifies the maximum width of a facet. If this option is not specified no upper limit on facet size will be applied.

Option RROPMF controls the faceted representation of the entity by considering the minimum size of the facet. The value in the option data specifies a 3-space facet width below which Parasolid may disregard the fact that facets don't meet the tolerance criteria specified via RROPST, RROPCT or RROPPT. Facets smaller than this dimension may still be produced, however. If this option is not specified, Parasolid will choose its own value for the minimum facet size based on the size of the face box. This avoids the problem whereby some portions of an entity may require arbitrary subdivision without achieving a satisfactory faceted representation.

Option RROPPT controls the facetted representation of the entity by considering the planarity of each facet. The first value in the option data is a distance tolerance. This is an upper bound on the distance from the facet to the facet mid-plane. The corners of the facet define the mid-plane which takes their average normal and passes through their center of gravity. The second value is an angular tolerance. This is defined as the ratio of the maximum separation between facet and mid-plane to maximum width over a facet. If this option is not specified the facets will be planar within modelling resolution.

If option RROPFP is specified, RRFCET will not generate facets for faces of solid bodies which can be quickly identified as back-facing in the given perspective view. This option is available in order to enhance performance; it does not guarantee to inhibit faceting of all back-facing faces.

If option RROPFI is specified, RRFCET will not generate facets for faces of solid bodies which can be quickly identified as back-facing in the given view from infinity. Only the view direction is taken from the view matrix. This option is available in order to enhance performance; it does not guarantee to inhibit faceting of all back-facing faces.

If option RROPTS is specified, RRFCET will output strips of triangular facets whenever possible. The option data value supplied with this option is the maximum number of facets in each strip. This value must be supplied and should be at least 2. With this option supplied the facets will always be triangular and therefore the maximum number of facet sides supplied with the option RROPFS will be ignored.

If option RROPIL is specified, RRFCET will make no attempt to trim the facetted representation of the surfaces to the specified loops. The loops must not be external loops of faces. This option is intended to be used to allow the facetting of sheet body faces to cover holes in the sheet by specifying the loops of the holes with this option.

If option RROPSD is specified, RRFCET will increase the density of facets over regions of the body which are silhouettes if the body is viewed with a parallel view in the direction sp[ecified by this data. The final data elements control this increase of the facet density.

Only one of RROPVM, RROPFP or RROPFI may be selected.

RRHIDL - Generate hidden line data

Receives

```
<KI_int_nitems>            *nopts          number of options in 'iopts'
KI_cod_rrop                 iopts[nopts]   rendering options
<KI_tag_list_dbl>           optdta[nopts]  option data lists
KI_tag_list_entity         *entys          entities to render
<KI_tag_list_<transform>>  *transf         entity transforms
KI_dbl_view_mx              vmatrx[16]     viewing transform matrix
```

Returns

```
KI_cod_error               *ifail          failure indicator
```

Specific Errors

```
KI_abort_from_go         Rendering aborted by GO
KI_bad_view_mx           Invalid view matrix
KI_wrong_transf          Transform contains scaling or reflection
KI_not_same_length       Entity and transform lists are of different
                         lengths
KI_wrong_entity_in_list  Entity is not a part
KI_bad_option_data       Option data incorrect for specified options
```

Description RRHIDL outputs, through the Graphical Output interface (GO), data for a picture of the entities, distinguishing visible from invisible portions.

RRHIDL can only render assemblies and complete bodies; it cannot render individual faces or edges. Visible edges and silhouette lines are always output.

Transforms may only contain translation and rotation components. Reflections, scales and shears are not allowed.

Other data output is selected by options in 'iopts'. For some options the output is further controlled by data in a real list whose tag is passed in the corresponding entry in 'optdta'. If no data is required for an option a null tag should be passed.

Options accepted are:-

Option	Contents of option data
RROPIV: InVisible	none
RROPDR: DRafting	none
RROPTR: TRansform	none
RROPIS: Image Smoothness	none
RROPDS: Drafting/Smooth edges behavior	none
RROPIN: INternal edges	none
RROPPC: B-Curve (Bezier)	none
RROPNC: B-curve (Nurbs)	none
RROPPS: Perspective	none
RROPRG: ReGional-data	none
RROPRA: Regional-Attrib	none
RROPPH: Planar-Hatch	(1) Planar hatch spacing in model units (2,3,4) Normal direction of hatch planes (5,6,7) Plane coordinate (omissible)
RROPRH: Radial-Hatch	(1) Hatch spacing in radians around 'spine' (2) Hatch spacing in model units along 'spine' (3) Hatch spacing in radians about center
RROPPA: PAra-hatch	(1) Hatch spacing along u (2) Hatch spacing along v
RROPCT: Curve Tolerance	(1) Chord tolerance in model units (2) Maximum chord length in model units (3) Angular tolerance in radians
RROPHR: Hierarchical	none
RROPHN: Hierarchical, no geometry	none
RROPHP: Hierarchical, parametrised	none
RROPVP: Viewport	(1,2,3) - Center of viewport in model coordinates (4,5,6) - 1st axis of viewport in model coordinates (7,8,9) - 2nd axis (10,11,12) - 3rd axis (13) - Length of viewport along 1st axis in model units (14) - Length along 2nd axis (15) - Length along 3rd axis

Option RROPIV causes data to be output for 'hidden' lines, so that for example, they can be rendered in a dotted line-style. By default, data is only output for visible lines.

Option RROPDR also causes data to be output for 'hidden' lines, but it distinguishes between lines which are hidden by other lines and those which are just obscured by other faces of the body, so that, for example, a plot of the image can be made in which no lines are drawn over the top of other lines. This option is not available for perspective views (i.e. it cannot be used with the option RROPPS). The performance will be downgraded if this option is turned on.

Option RROPTR is required if any of the entities is to be rendered in a transformed position (e.g. as part of an assembly). If it is not specified, the contents of 'transf' are ignored, and entities are rendered in their local coordinate system.

Option RROPIS causes the output data for an edge to specify whether or not the edge is smooth (i.e. the faces have the same tangent surface at the edge) or smooth but coincident with a silhouette line (which is not output). The calling program may then choose to omit smooth edges from the final drawing. If RROPIS is not specified, the smoothness parameter will be "unknown" for all edges.

Option RROPDS controls the behavior of smooth edges in drafting mode. The calling program should select this option if it is intending to omit smooth edges which are not coincident with silhouettes (those tagged CODSMO) from the image. When the option is selected, lines hidden by such edges will be considered to be obscured by the faces of the smooth edge rather than by the edge itself, i.e. they will be tagged as CODDRV. If this option is not selected, then CODSMO lines will obscure other lines in the picture, leading to apparent errors in the image if the CODSMO lines are not drawn. It is assumed that smooth edges coincident with silhouettes (those tagged CODSMS) are always included in the image, hence they will obscure other lines regardless of whether this option is selected.

Option RROPIN causes the output data for an edge to specify whether or not the edge is internal to a surface; i.e. the faces on either side of the edge lie on the same surface. This will be the case if the edge is an internal one between patches on a B-surface, or is mergeable. The output data will also specify whether the edge is coincident with a silhouette line (which is not output), or whether the edge is part of the body silhouette. The calling program may then choose to omit internal edges from the final drawing. If RROPIN is not specified, the internal edge parameter will be "unknown" for all edges.

Option RROPPC causes the data for a B-Curve (type TYCUPA) to be output in Bezier form.

Option RROPNC causes the data for a B-Curve (type TYCUPA) to be output in NURBS format.

The default, if neither RROPPC or RROPNC is supplied, is to output the data for B-Curves as a poly-line.

Note that at most one of RROPPC and RROPNC may be selected.

Option RROPPS should be specified if a perspective view is required; by default the system will produce a view from infinity, taking only the view direction from 'vmatrx'.

Option RROPRG causes regional data to be produced for all visible edges and silhouettes.

Option RROPRA causes regional data to be produced for all visible edges and silhouettes adjacent to any face with the regional-data (TYSARG) attribute attached.

Options RROPPH, RROPRH and RROPPA cause hatch lines to be produced for various types of face as defined in the documentation for RRVIND.

Option RROPCT controls the representation of curves as polylines. A curved edge is approximated by a number of straight lines called chords. The first value in the option data is a chord tolerance. This is an upper bound on the distance from each chord to the curve it is approximating. The second value is a chord limit. This is an upper bound on the length of a chord used in the approximation of a curved edge. The third value is an angular tolerance. This limits the angular error between a curve and a chord used in its approximation. This is an upper bound on the sum of the two angles formed between the chord and the curve tangent at each chord end. If this option is not specified a default angular tolerance is used.

Bodies are always output hierarchically: for each body to be processed a hierarchical segment of type body is opened, all the segments in that body are output, then the hierarchical segment is closed.

If the option RROPHR is specified then edges, silhouettes and hatch-lines will be output hierarchically. Hierarchical segments of type edge, silhouette and hatchline are output/closed by GOOPSG/GOCLSG respectively. The non-hierarchical segments enclosed by GOOPSG and GOCLSG comprise a geometry segment which gives the geometry of the entire edge/silhouette/hatch-line, and a visibility segment which gives the points at which the visibility changes and the visibility codes for the edge/silhouette/hatch-line between these points.

If the option RROPHN is specified then the output will be the same as for RROPHR but the geometry segment will not be output.

If the option RROPHP is specified then the output will be the same as for RROPHR but visibility segments will be parametrised (i.e. of type SGTPVP) when the geometry segment is a polyline. See the Downward Interfaces and Functional Description manuals for more information on parametrised visibility segments.

If the option RROPVP is supplied then the system will attempt only to render those bodies/faces which are inside or partly inside the viewport supplied.

The viewport is a cuboid region of model space and should be supplied in the option data as a real list consisting of a center, three axes, and lengths in model units along these axes. The axes must be orthogonal but need not be normalized.

While all faces that are inside or partly inside the viewport will be rendered, no attempt is made to trim the output to the boundaries of the viewport, and bodies or faces lying entirely outside, but close to it may be rendered.

Note: When this option is used with RRHIDL, it is possible to make visible faces that would not otherwise be so by excluding from the viewport the parts that would obscure them.

RRVDEP - Generate view dependent rendering data

Receives

```
<KI_int_nitems>            *nopts          number of options in 'iopts'
KI_cod_rrop                 iopts[nopts]   rendering options
<KI_tag_list_dbl>           optdta[nopts]  option data lists
KI_tag_list_entity         *entys          entities to render
<KI_tag_list_<transform>>  *transf         entity transforms
KI_dbl_view_mx              vmatrx[16]     viewing transform matrix
```

Returns

```
KI_cod_error               *ifail          failure indicator
```

Specific Errors

```
KI_abort_from_go          Rendering aborted by GO
KI_bad_view_mx            Invalid view matrix
KI_wrong_transf           Transform contains scaling or reflection
KI_not_same_length        Entity and transform lists are of different
                          lengths
KI_wrong_entity_in_list   Entity is not a part, face or edge
KI_bad_option_data        Option data incorrect for specified options
```

Description RRVDEP outputs, through the Graphical Output interface (GO), data for a picture of the entities which is dependent on the viewing angle and distance (e.g. silhouettes); view independent data is not output.

RRVDEP can render assemblies, complete bodies and/or individual faces. Edges are ignored.

Transforms may only contain translation and rotation components. Reflections, scales and shears are not allowed.

The data output is selected by options in 'iopts'. For some options the output is further controlled by data in a real list whose tag is passed in the corresponding entry in 'optdta'. If no data is required for an option a null tag should be passed.

Options accepted are:-

Option	Contents of option data
RROPSI: SIlhouette-data	none
RROPTR: TRansform	none
RROPPS: Perspective	none

RROPCT: Curve Tolerance	(1) Chord tolerance in model units (2) Maximum chord length in model units (3) Angular tolerance in radians
RROPVP: Viewport	(1,2,3) Center of viewport in model coordinates (4,5,6) 1st axis of viewport in model coordinates (7,8,9) 2nd axis (10,11,12) 3rd axis (13) Length of viewport along 1st axis in model units (14) Length along 2nd axis (15) Length along 3rd axis

Option RROPSI causes silhouette lines to be output. If not present no data will be produced by RRVDEP. Silhouettes coincident with real edges are not output.

Option RROPTR is required if any of the entities is to be rendered in a transformed position (e.g. as part of an assembly). If it is not specified, the contents of 'transf' are ignored, and entities are rendered in their local coordinate system.

Option RROPPS should be specified if a perspective view is required; by default the system will produce a view from infinity, taking only the view direction from 'vmatrx'.

Option RROPCT controls the representation of curves as polylines. A curved edge is approximated by a number of straight lines called chords. The first value in the option data is a chord tolerance. This is an upper bound on the distance from each chord to the curve it is approximating. The second value is a chord limit. This is an upper bound on the length of a chord used in the approximation of a curved edge. The third value is an angular tolerance. This limits the angular error between a curve and a chord used in its approximation. This is an upper bound on the sum of the two angles formed between the chord and the curve tangent at each chord end. If this option is not specified a default angular tolerance is used.

If the option RROPVP is supplied then the system will attempt only to render those bodies/ faces which are inside or partly inside the viewport supplied.

The viewport is a cuboid region of model space and should be supplied in the option data as a real list consisting of a center, three axes, and lengths in model units along these axes. The axes must be orthogonal but need not be normalized.

While all faces that are inside or partly inside the viewport will be rendered, no attempt is made to trim the output to the boundaries of the viewport, and bodies or faces lying entirely outside, but close to it may be rendered.

RRVIND - Generate view independent rendering data

Receives

```
<KI_int_nitems>          *nopts        number of options in 'iopts'
```

```
KI_cod_rrop                    iopts[nopts]   rendering options
<KI_tag_list_dbl>              optdta[nopts]  option data lists
KI_tag_list_entity            *entys          entities to render
<KI_tag_list_<transform>>     *transf         entity transforms
```

Returns

```
KI_cod_error                  *ifail          failure indicator
```

Specific Errors

```
KI_abort_from_go          Rendering aborted by GO
KI_wrong_transf           Transform contains scaling or reflection
KI_not_same_length        Entity and transform lists are of different
                          lengths
KI_wrong_entity_in_list   Entity is not a part, face, edge or geometry.
KI_bad_option_data        Option data incorrect for specified options
```

Description RRVIND outputs, through the Graphical Output interface (GO), data for a picture of the entities which is independent of the viewing angle and distance.

RRVIND can render assemblies, bodies, faces, edges, B-Curves and B-Surfaces, foreign geometry and offsets of B-surfaces and foreign geometry.

Transforms may only contain translation and rotation components. Reflections, scales and shears are not allowed.

The data output is selected by options in 'iopts'. For some options the output is further controlled by data in a real list whose tag is passed in the corresponding entry in 'optdta'. If no data is required for an option a null tag should be passed.

Options accepted are:-

Option	Contents of option data	
RROPED: EDge-data		
RROPTR: TRansform	none	
RROPSM: SMooth-edges		
RROPIE: Internal Edge	none	
RROPPC: B-Curve (Bezier)	none	
RROPNC: B-curve (Nurbs)	none	
RROPPH: Planar-Hatch	(1)	Planar hatch spacing in model units
	(2,3,4)	Normal direction of hatch planes
	(5,6,7)	Plane coordinate (omissible)
RROPRH: Radial-Hatch	(1)	Hatch spacing in radians around 'spine'
	(2)	Hatch spacing in model units along 'spine'
	(3)	Hatch spacing in radians about center
RROPPA: PAra-hatch	(1)	Hatch spacing along u
	(2)	Hatch spacing along v

RROPUB: Unfixed-Blends	(1)	Rendering style:-
	1 =>	Draw as specified in model
	2=>	Draw blend boundaries
	3=>	Draw blend boundaries and rib lines
	(2)	Rib spacing in model units (only required for style 3)
RROPCT: Curve Tolerance	(1)	Chord tolerance in model units
	(2)	Maximum chord length in model units
	(3)	Angular tolerance in radians
RROPVP: Viewport	(1,2,3)	Center of viewport in model coordinates
	(4,5,6)	1st axis of viewport in model coordinates
	(7,8,9)	2nd axis
	(10,11,12)	3rd axis
	(13)	Length of viewport along 1st axis in model units
	(14)	Length along 2nd axis
	(15)	Length along 3rd axis

When rendering B-Curves and surfaces, only the following options are relevant:-

RROPTR, RROPPC, RROPNC, RROPPA and RROPCT.

The default for these types is to render the boundary, for a surface, and the curve for a curve. Therefore, in contrast to the other entity types, output will be generated even in the absence of any accompanying options.

Note: Only options RROPED and RROPTR are allowed on parts (or items from parts) whose tolerance is greater than Parasolid's. The same also applies to parts containing SP-curves.

Option RROPED causes output of data for edges in the model.

Option RROPTR is required if any of the entities is to be rendered in a transformed position (e.g. as part of an assembly). If it is not specified, the contents of 'transf' are ignored, and entities are rendered in their local coordinate system.

Option RROPSM causes the output data for an edge to specify whether or not the edge is 'smooth' (i.e. the faces have the same tangent surface at the edge); the calling program may then choose to omit smooth edges from the final drawing. If RROPSM is not specified, the smoothness parameter is returned "unknown" for all edges.

Option RROPIE causes the output data for an edge to specify whether or not the edge is internal to a surface; i.e. the faces on either side of the edge lie on the same surface. This will be the case if the edge is an internal one between patches on a B-Surface, or is mergeable. The calling program may then choose to omit internal edges from the final drawing. If RROPIE is not specified, the internal edge parameter will be "unknown" for all edges.

Option RROPPC causes the data for a B-Curve (type TYCUPA) to be output in Bezier form.

Option RROPNC causes the data for a B-Curve (type TYCUPA) to be output in NURBS format.

The default, if neither RROPPC or RROPNC is selected, is to output the data for B-Curves as a poly-line. These options both apply to derived B-Curves as well. For example, hatch and boundary lines on B-Surfaces.

> **Note:** At most one of RROPPC and RROPNC may be selected.

Option RROPPH causes planar hatch lines to be output. If no option data is given faces which have a planar hatching attribute set will be hatched as specified by that attribute. If a hatch specification is given in the option data all faces not otherwise hatched by use of the RROPRH and RROPPA options will be hatched according to that specification.

Option RROPRH causes radial hatch lines to be output for faces with standard curved surfaces (cylindrical, conical, spherical, toroidal) or blending surfaces. If no option data is given those faces which have a radial hatching attribute set will be hatched as specified by that attribute. If a hatching specification is given in the option data all faces with standard curved surfaces or blending surfaces will be hatched according to that specification.

The values given in the option data are interpreted according to the type of surface to which they are applied as follows:

- Cylindrical and Conical surfaces:

  The first value controls the spacing of straight hatch lines running along the surface parallel to the axis. Such lines would be generated by a plane rotating in steps about the axis by the given angle. This angle must be in the range 0 to two pi.

  The second value controls the spacing of circular hatch lines running around the surface perpendicular to the axis. Such lines would be generated by a plane stepping along the axis by the given distance.

  The third value is not significant.

- Spherical surfaces:

  The hatching of spherical surfaces is related to an axis passing through the center of the surface and parallel to the Z axis.

  The first value controls the spacing of the 'longitudinal' hatch lines, as would be generated by a plane rotating by the given angle about the axis of the surface. This angle must be in the range 0 to two pi.

  The third value controls the spacing of the 'latitudinal' hatch lines, as would be generated by a cone (apex at center of surface, axis parallel to surface axis) whose half angle is incremented by the given value. This value must be in the range 0 to two pi.

  The second value is not significant.

- Torus:

The first value controls the spacing of circular hatch lines center on the axis of the torus, as would be generated by a cone (axis parallel to torus axis, passing through spine of torus) whose half angle is incremented by the given value. This value must be in the range 0 to two pi.

The second and third values control the spacing of circular hatch lines centered on the spine of the torus, as would be generated by a plane rotating about the axis of the torus. The second value specifies a distance along the spine whereas the third value specifies an angle about the axis. If both second and third values are given the second value is ignored. If a third value is given, it must be in the range 0 to two pi.

- Edge-Blend surfaces:

  The second value controls the spacing of 'circular' hatch lines running around the surface perpendicular to its spine. These are equivalent to the circles generated by this value for cylindrical surfaces. Such lines would be generated by a plane stepping along the surface spine by the given distance.

  The first and third values are not significant.

Option RROPPA causes parametric hatch lines to be output, which correspond to the locus of points of constant parameter value on the surface. The lines may be output on faces with any type of underlying surface, though longitudinal hatch lines on rolling ball blends (i.e. lines of constant v parameter) are not supported.

If no option data is given, only those faces which have a hatching attribute set will be hatched as specified by that attribute. If a hatching specification is given in the option data, all faces will be hatched according to that specification. Two values are required for the option data: the first corresponds to the u parameter spacing, and the second the spacing in v. Both values must be positive, and should lie within the parameter range for the surface, which can be output using ENSUPA.

In both the RROPRH and RROPPA options the hatch lines generated are controlled by two significant values for any given surface. If either value is zero hatch lines are not generated for that case. If both values are non-zero the face is cross hatched.

Option RROPUB is required if rendering is to take account of unfixed blending surfaces; by default these are ignored. If RROPUB is specified with no option data, unfixed blends will be rendered according to the attributes held in the model; alternatively, option data can be used to specify how rendering is to be done.

The edges of faces adjacent to the unfixed blend will be clipped to the blend boundaries but hatch lines on such faces are not affected.

This option is ignored for general bodies, however - i.e. unfixed blends on general bodies are not rendered.

Option RROPCT controls the representation of curves as polylines. Note that circles, lines and ellipses are always output exactly (and not as polylines). A curved edge is approximated by a number of straight lines called chords. The first value in the option data is a chord tolerance. This is an upper bound on the distance from each chord to the curve it is approximating. The second value is a chord limit. This is an upper bound on the length of a chord used in the approximation of a curved edge. The third value is an angular tolerance. This limits the angular error between a curve and a chord used in its approximation. This is an upper bound on the sum of the two angles formed between the chord and the curve tangent at each chord end. If this option is not specified a default angular tolerance is used.

If the option RROPVP is supplied then the system will attempt only to render those bodies/faces which are inside or partly inside the viewport supplied.

The viewport is a cuboid region of model space and should be supplied in the option data as a real list consisting of a center, three axes, and lengths in model units along these axes. The axes must be orthogonal but need not be normalized.

While all faces that are inside or partly inside the viewport will be rendered, no attempt is made to trim the output to the boundaries of the viewport, and bodies or faces lying entirely outside, but close to it may be rendered.

SAVMOD - Save model in archive

Receives

KI_tag_part	*part	part to transmit
KI_int_nchars	*keylen	length of key
KI_chr_key	key[keylen]	key of part

Returns

KI_cod_error	*ifail	failure indicator

Specific Errors

KI_applio_not_registered	application i/o functions not registered
KI_disc_full	disc full
KI_file_access_error	error writing or closing the transmit file
KI_cant_open_file	error opening the transmit file
KI_schema_access_error	error opening, closing or writing the schema file
KI_modified_sub_part	part has modified sub-part
KI_part_not_isolated	part not isolated
KI_already_saved	part is already stored in archive
KI_key_in_use	key already used in archive, can't transmit modified part with same key
KI_bad_key	invalid key syntax

Description SAVMOD transmits 'part' and its true-sub-parts into the archive with the given 'key'.

The part must be new or modified (by definition stored, anonymous and unloaded parts are already in the archive).

Whether parts are transmitted in text or in binary depends on the value of the SLIPBT interface parameter (see SEINTP for details).

If 'part' is modified 'key' must not match the key already attached to the part. In fact if the key matches the key of any other part in memory KI_key_in_use will be returned in 'ifail'. This error code may also be returned if the key is already used in the archive.

We define the true-sub-parts of a part P to be all sub-parts of P which can be reached without encountering a keyed (i.e. stored, unloaded or modified) part.

The part and all true-sub-parts of the part are transmitted into the archive under the same key. When transmitted the part is changed to stored and all true-sub-parts are changed to anonymous. For example (the parts marked * are the true-sub-parts of P, and ST represents a stored part and its true-sub-parts):

```
      Before transmit of P                  After transmit of P

          P->md                      P->st                          ST
           /  \                       /  \                         /  \
        *nw    *nw                  an    an                     ST    ST
        / \    / \                 / \   / \          OR               / \
     *nw   *nw    ST             an   an    ST                       ST   ST
      /           / \            /          / \                       \   /
     ST          ST  ST         ST         ST  ST                      ST
                  \  /                      \  /
                   ST                        ST
```

If the part is to be transmitted it must be the root of an isolated sub-graph which references only stored or unloaded parts. If these conditions are not met the state changes would cause EITHER a new or modified part to become a sub-part of a stored part OR a stored part to become the root of a non-isolated sub-graph. For example:

Before transmit of P — After transmit P is stored but has modified sub-part and thus SAVMOD will fail returning KI_modified_sub_part in 'ifail'.

```
          P->md                            P->st
           /  \                             /  \
         nw    nw                         an    an
        / \   /                          /  \  /
      nw    md                         an    md
```

Before transmit of P — After transmit P is stored but is not root of isolated sub-graph and thus SAVMOD will fail returning KI_part_not_isolated in 'ifail'.

```
          P->md                            P->st
           /  \                             /  \
         nw    nw   md                    an    an   md
        / \   / \   /                    /     / \   /
      nw    nw    nw                   an    an    an
```

SAVSNP - Make a snapshot

Receives

```
KI_int_nchars    *nchars         number of characters in filename
KI_chr_filename   filnam[nchars] filename for snapshot
int              *histfl         unused: should be zero
```

Returns

```
KI_cod_error     *ifail          error code
```

Specific Errors

```
KI_bad_text_conversion invalid value for text output
KI_file_already_exists snapshot file of the given name already exists
KI_schema_access_error error opening, closing or writing the schema file
KI_file_access_error   error writing or closing snapshot file
KI_cant_open_file      cannot open snapshot file
KI_bad_filename        invalid filename
```

Description Saves the state of tag-memory in a snapshot file, so that it can later be restored via a call to GETSNP. Rollback information is not stored. Note that a snapshot file can only be restored by the same version of the modeler which generated it.

SCRIBE - Scribe line on face, a region or a wire body

Receives

```
KI_tag_topology    *topol  face, region or body to scribe curve on
KI_tag_curve       *curve  curve to scribe
KI_vec_position     startp start position
KI_vec_position     endp   end position
```

Returns

```
KI_tag_list_edge   *newedg list of new edges
KI_int_nitems      *nedges number of new edges
<KI_tag_list_face> *newfac list of new faces, if any
<KI_int_nitems>    *nfaces number of new faces
KI_cod_error       *ifail  failure indicator
```

Specific Errors

```
KI_missing geom          given face has no surface
KI_general_body          given region is from general body
KI_non_manifold          body would become general
KI_not_in_region         no new edges created
KI_bad_end_points        curve not defined between 'startp' and 'endp'
KI_not_on_curve          start or end point not on curve
KI_not_on_surface        curve does not lie on surface of face
KI_not_on_face           no new edges created
KI_crossing_edge         curve crosses edge
KI_crossing_face         curve crosses face
```

KI_unsuitable_entity	topology is not a face, a region, or a wire body
KI_bad_basis_surf	SP-curve basis surface differs from face surface
KI_invalid_geometry	curve does not pass checks
KI_not_in_same_partition	topol and curve are in different partitions

Description The given curve, between the start and end points, is inscribed on 'topol'.

If 'topol' is a face, each section of the curve that lies in the interior of the face gives rise to a new edge or edges: this may have the effect of dividing the face into several pieces, all but one of which will be new faces. The face must have a surface attached to it, or the ifail KI_missing_geometry will be returned.

If 'topol' is a region, each section of the curve that lies in the interior of the region gives rise to a new edge or edges. Under certain circumstances a new face will also be made; see below.

If 'topol' is a body, it must be a wire body or a minimal body and the curve will be scribed as if the region (which is unique in this case) of the body had been given.

If 'topol' is of any other type, the ifail KI_unsuitable_entity will be returned.

If the curve is a B-curve (TYCUPA) then it must be capable of passing the checks imposed by CHCKEN. The self intersection check is only performed if SLIPSI not zero (see SEINTP). The composite geometry checks are only performed if SLIPCO (see SEINTP) is set to 0.

A list of the new edges created will be returned in 'newedg'. If new faces are created, they are returned in the list 'newfac'; if no new faces are created, 'newfac' will be the null tag.

The start and end points must lie on the curve or the ifail KI_not_on_curve will be returned. If they are in the same position the complete closed curve is scribed. If they are not, the start point must precede the end point or the ifail KI_bad_end_points will be returned. If the curve is a trimmed curve the bounds are ignored, though valid vectors should still be supplied.

If a trimmed curve is supplied, it will be deleted and the underlying curve attached to the resulting edge or fin.

If local checking is on (see SEINTP), the result body is checked for edge-edge and edge-face inconsistencies between the new edges and the rest of the body. If any edge fails these checks one of the ifails KI_crossing_edge or KI_crossing_face will be returned.

Scribe may be used in the following cases:

- Region of a minimal body or a minimal body):

  If the start and end points differ, an open wire body will result. If they are the same, the result depends on the setting of SLIPGT (see SEINTP):

  - if SLIPGT is zero, the result will be a sheet body with a single face without a surface attached; the face will lie on the left of the new edge, which will go in the direction of the curve.
  - if SLIPGT is one, the result will be a closed wire body.

The position of the point of the single vertex of the body is irrelevant.

- Region of a wire body or a wire body:
  - If SLIPGT is zero:

    The start and end points must be different, and one or both must be an end vertex of the (open) wire or the ifail KI_non_manifold will be returned. If it joins the end vertices of a wire, then the result is a sheet body with a single face without a surface attached; the face will lie on the left of the new edge, which will go in the direction of the curve. The new edges may not touch or cross the interior of an edge of the wire or the ifail KI_crossing_edge will be returned. The new edges may not touch or cross an interior vertex of the wire or the ifail KI_non_manifold will be returned.

  - If SLIPGT is one:

    The bounded curve may or may not cross the existing vertices of the body. It may not be coincident with the existing edges of the body, or cross them except at the vertices or the ifail KI_crossing_edge will be returned.

- This operation will in some cases (ie unless the rules which apply when SLIPGT is zero are followed) result in a non-manifold or disconnected body; if it does, the body type will change to general.
- Any region of a sheet, solid, or general body:

  SLIPGT must be set to one, or one of the ifails KI_non_manifold or KI_general_body will be returned.

  If the curve, between the start and end points, does not lie in the region, the ifail KI_not_in_region will be returned.

  The bounded curve may or may not cross the existing vertices of the body. It may not coincide with existing edges or faces of the body, or cross them except at the vertices or one of the ifails KI_crossing_edge or KI_crossing_face will be returned.

  One or more wireframe edges will be created in the given region. If the region is in a sheet or solid body, this operation will always result in a non-manifold or disconnected body, and so the body type of a sheet or solid body will change to general. If the region is in a general body the operation may result in a wire body, in which case the body type will change to wire. The operation may involve the creation of a new shell in the given region, or even the merging of two of its existing shells.

- A non-rubber face of a sheet, solid or general body:

  The curve must lie in the surface of the face or the ifail KI_not_on_surface will be returned. At least one new edge must be created by the scribe, or the ifail KI_not_on_face will be returned. If new faces are created, the old face will lie on the right of one of the scribed edges, which will all go in the direction of the curve.

Scribing SP-curves is supported, providing they reference the surface of the face being scribed or a copy of this surface.

SEBBCO - Set bulletin board controls

Receives

```
<KI_int_nitems>  *nents          length of 'ents' array
KI_cod_ty         ents[nents]    entity types to be bulletined
KI_tag_list_int   events[nents]  corresponding events to record
<KI_int_nitems>  *nopts          no of control options
KI_cod_bbop       opts[nopts]    control options
```

Returns

```
KI_cod_error     *ifail          error code
```

Specific Errors

```
KI_cod_bbev                  invalid event code
KI_bad_selection_code        invalid event code
KI_bad_entity_event_comb     event/entity mismatch
KI_bad_type                  invalid type token supplied
KI_no_user_fields            can't bulletin user fields when there are
                             none
KI_bad_state_combn           option inconsistent with state of bulletin
                             board
```

Description SEBBCO controls how information, if any, is recorded by the bulletin board and turns it on and off. Control is specified by means of control options held in the array 'opts'. Information on when an entity should be recorded on the bulletin board is specified by means of data held in arrays 'ents' and 'events'.

Controlling when entities are to be recorded is achieved by associating with an entity type a list of event tokens (taken from the range BBEV00).

For a given entity type the event tokens instruct Parasolid to record an entity (of the given type) on the bulletin board when a specified event occurs to that entity.

Event Token	Interpretation
BBEVCR	Record on creation
BBEVDE	Record on deletion
BBEVCO	Record when copied
BBEVTR	Record when transferred
BBEVME	Record when merged
BBEVSP	Record when split
BBEVTF	Record when transformed
BBEVAC	Record when attribute owned by entity changes
BBEVCH	Record when changed

The arrays 'ents' and 'events', each of size 'nents', hold corresponding data; each entry in 'events' is a list indicating which events are to be recorded for entities of the type whose code is given in the corresponding position in the array 'ents'.

Certain event types cannot apply to certain entity types (for instance, a point (type TYTOPT) is never merged or split (event types BBEVME and BBEVSP) and some are not recorded for certain entities (for instance, a body (type TYTOBY) is not recorded as changed (event type BBEVCH). A request for the bulletin board to record an event for an entity for which it is inappropriate for either of these reasons is an error and will result in KI_bad_entity_event_comb being returned in 'ifail'. The valid events for entities are indicated by crosses in the following table; a blank space in the table indicates that the event code may not be supplied for the given entity types.

	BBEV..								
	CR	**DE**	**CO**	**TR**	**ME**	**SP**	**TF**	**AC**	**CH**
TYTO.. BY, AS	X	X	X		X	X	X	X	
TYTO.. IN, SH, FA, LO, ED, VX, RG	X	X	X	X	X	X	X	X	X
TYADFE	X	X	X	X	X	X	X	X	X
TYGE.. SU, CU, PT, TF	X	X	X	X			X	X	X
TYADAD	X								

In the array 'opts' codes specifying further detail about what information the bulletin board is to record may be supplied. These are specified by tokens from the range BBOP00. At present there is only one option which can take three possible values. These are:

- Switch bulletin board off (token BBOPOF)
- Switch bulletin board on to record tags only (token BBOPON)
- Switch bulletin board on to record tags and user fields (token BBOPUF)

If more than one code is supplied in 'opts', the codes will be applied successively, starting with 'opts[0]' and continuing to 'opts['nopts'-1]'.

If user-fields are not being allocated (i.e. STAMOD was called with 'usrfld' = 0) and option code BBOPUF is received then KI_no_user_fields is returned in 'ifail'. If an option code is received which clashes with, or is meaningless given, the current state of the bulletin board, KI_bad_request_code is returned in 'ifail'. When the board is off, only a request to switch it off is invalid. When the board is on, whether with user fields or not, it is possible to switch it off. The only other legitimate option is when the board is on but not recording user fields and there is no information held in it; in this case, it is permissible to supply code BBOPUF to switch on recording of user fields. The bulletin board may be empty either because no events have occurred since it was switched on, or because it has been emptied in the course of reading it.

SECTBY - Section bodies

Receives

```
KI_tag_list_body  *bodies  body or list of bodies to be sectioned
KI_tag_surface    *surfac  sectioning surface
```

Returns

```
KI_tag_list_body  *front   bodies in front of surface
KI_tag_list_body  *back    bodies behind surface
KI_tag_list_face  *newfas  new faces
<KI_int_nitems>   *nfaces  number of faces in 'newfas'
KI_cod_error      *ifail   failure code
```

Specific Errors

```
KI_invalid_bodies     Boolean failure or invalid bodies
KI_partial_coi_found  Boolean failure due to partial coincidence
KI_wrong_surface      Surface not plane, cylinder
KI_wire_body          A body is a wire body
KI_missing_geom       A body has incomplete geometry
KI_non_manifold       Non-manifold result
KI_general_body       General body
```

Description The given bodies are sectioned with the given surface, which may be planar or cylindrical.

When the section surface intersects with a body, this body will be split (by the section surface) into two or more parts.

The resulting bodies are sorted into two lists: those in front of the section surface (on the side to which the surface normal points) and those behind it. They are returned in the lists 'front' and 'back' respectively. A list of the new faces created is also returned, in the list 'newfas'. All these faces will lie in the sectioning surface.

For each body in 'bodies', all assemblies that instanced the body will now instance all the pieces of the body.

Any bodies unaffected by the sectioning operation will be left unchanged, but will be put into one or other of the 'front' or 'back' lists as appropriate.

After a successful sectioning operation, any of the lists 'front', 'back' and 'newfas' may be returned as empty lists, but not null.

This function does not support general bodies.

SEINTP - Set interface parameter

Receives

```
KI_cod_slip       *pnum     parameter code
int               *ival     integer value of parameter
double            *rval     real value of parameter
```

Returns

```
KI_cod_error      *ifail    failure indicator
```

Specific Errors

```
KI_tag_limit_out_of_range Tag limit lies out of range
KI_bad_value              Continuity check code lies out of range;
                          Binary/text snapshot control code lies outside
                          range; Binary transmission control code lies
                          outside range; Bulletin board control code
                          lies outside range; Bad code
KI_roll_forward_fail      Rollback already active
```

KI_bad_rollfile_size	Size given for rollback file too large or too small
KI_cant_open_file	Failed to open rollback file
KI_no_user_fields	Cannot bulletin user fields of zero length
KI_journal_not_open	Journalling was not requested in call to STAMOD

Description The value of 'pnum' is used to indicate which parameter to set and thereby which value ('ival' or 'rval') to use. At present this routine only receives integer parameters.

The valid values of 'pnum' and the corresponding parameters are:

`pnum'	receive	parameter
SLIPCH	`ival'	Flag indicating whether other KI routines and PK routines should test their parameters. Passing non-zero 'ival' will turn it on. This flag is set to non_zero in STAMOD. When this is off dead and null tags are still trapped by KI routines but not by PK routines.
SLIPJO	`ival'	Flag indicating whether calls to KI routines will produce output in the jour nal file. Passing non-zero 'ival' will turn it on. This flag is set in STAMOD according to the value of the received argument. If journalling was not requested in STAMOD ('kijon' set to false) then 'ifail' KI_journal_not_open will be returned from SEINTP if it is called with 'pnum' set to SLIPJO regardless of the value of 'ival'.
SLIPBB	`ival'	Flag indicating whether the Kernel maintains a bulletin board. The defined values of 'ival' are: 0 - no bulletin board maintained 1 - bulletin board is emptied and will be maintained for tags only 2 - bulletin board is emptied and will be maintained for tags and user fields This flag is set to zero in STAMOD. If 'ival' is 1 or 2 then a default bulletin board state is initialized. This will instruct the bulletin board to record information about faces, edges and vertices (types TYTOFA, TYTOED and TYTOVX) for events create, delete and change (event types BBEVCR, BBEVDE and BBEVCH) and bodies and assemblies (types TYTOBY and TYTOAS) for events create and delete. See SEBBCO for more information.
SLIPRB	`ival'	Integer flag indicating if logging for rollback is to be enabled. This must be set in order that roll-back marks may be set. Passing non-zero 'ival' will turn it on. The absolute value of 'ival' is the maximum size, in bytes, of the rollback file. This size should be a few times the size of the model and typically will be sev eral Mbytes. Routine FFOPRB may allocate a file of smaller size, and the actual size may be discovered by a call to OUINTP. This flag is set to zero in STAMOD.

SLIPRF	\`ival'	Integer flag indicating whether roll forward is to be enabled. This must be set while rollback is switched off. If 'ival' is 1 rolling forward will be enabled when rollback is enabled. If 'ival' is 0 then less information needs to be stored in the rollfile. This flag is set to zero in STAMOD.
SLIPBT	\`ival'	Controls the use of binary transmission. The defined values of 'ival' are: 1 text Receive, text Transmit 2 binary Receive, text Transmit 3 text Receive, binary Transmit 4 binary Receive, binary Transmit 5 text Receive, neutral Transmit 6 binary Receive, neutral Transmit 7 applio Receive, text Transmit 8 applio Receive, binary Transmit 9 applio Receive, neutral Transmit 10 applio Receive, applio Transmit 11 text Receive, applio Transmit 12 binary Receive, applio Transmit Values outside this range return KI_bad_value. If ival is 5, 6 or 9, Parasolid will write machine-independent binary files. These can be read in as normal binary transmit files, i.e. with ival set to 2, 4, 6 or 12. This flag is set to 1 in STAMOD.
SLIPSN	\`ival'	Controls whether snapshot files are binary or text. The defined values of 'ival' are: 1 text get snapshot, text save snapshot 2 binary get snapshot, text save snapshot 3 text get snapshot, binary save snapshot 4 binary get snapshot, binary save snapshot Values outside this range return KI_bad_value. This flag is set to 4 in STAMOD.
SLIPLC	\`ival'	Flag indicating whether the Kernel will perform local checks of geometry and topology after a local modification is made to the model by CRFASU, CRSO FA, DELFAS, RMFASO, SWEENT, SWIENT, TWSUFA, TWEFAC, TAPFAS or BLEFIX. Passing non-zero 'ival' will turn local checking on. This flag is set to non-zero in STAMOD.
SLIPDC	\`ival'	Flag indicating whether limited checks on the consistency of geometry at tached to a model with ATTGEO and neighboring geometry will be per formed. Passing non-zero 'ival' will turn data checking on. This flag is set to zero in STAMOD.

SLIPUF	`ival'	Controls whether user-field information is received from archived parts. Normally this should be set to 1 so that models are received with the same user-field values as when they were transmitted. However, there are situa tions where this is undesirable (for example, when receiving a model with a different user-field length). In this case SLIPUF should be set to zero so that user-fields in parts received subsequently will be set to zero. See GETMOD, page , for further details. This flag is set to 1 in STAMOD.
SLIPSI	`ival'	Flag indicating whether the Kernel will perform self intersection checks on geometry during those operations which require the geometry to pass the checks imposed by CHCKEN. Passing non-zero 'ival' will turn self intersec tion checking on. This flag is set to non-zero in STAMOD.
SLIPCO	`ival'	Flag indicating whether the Kernel will perform composite geometry checks on B-curves and B-surfaces when determining whether the geometry can be modeled as a single topological entity. The defined values of `ival are : 0 Perform all composite geometry checks 1 Perform no composite geometry checks Values outside this range return KI_bad_value. This flag is set to zero in STAMOD.
SLIPTL	`ival'	Integer flag indicating if tag limitation is enabled. Passing a non-zero 'ival' will turn it on, a zero 'ival' will turn it off. A non-zero value of 'ival' less than an existing tag will return KI_tag_limit_out_of_range. If non-zero, the modeler will check all tags allocated against an upper bound equal to the absolute value of 'ival'. An operation that would otherwise result in the allocation of tags exceeding 'ival' will fail, returning KI_tag_limit_exceeded. This flag is set to zero in STAMOD.
SLIPGS	`ival'	Flag indicating whether generated surfaces can be created during a call to SWEENT or SWIENT. If non-zero, generated surfaces can be created dur ing a call to SWEENT or SWIENT. If zero, B-surfaces will be created instead. This flag is set to zero in STAMOD.
SLIPGT	`ival'	Flag indicating whether general bodies are to be legally returned by KI func tions - principally boolean operations. With SLIPGT set to zero, general bodies will not be returned and the operations will fail with the same ifail as at previous versions i.e. KI_non_manifold. With SLIPGT set to one, general bodies will be returned as valid results. This flag is set to zero in STAMOD.

SEMODP - Set modeller parameter

Receives

```
KI_cod_slmp        *pnum     parameter code
int                *ival     integer value of parameter
double             *rval     real value of parameter
```

Returns

```
KI_cod_error      *ifail    failure indicator
```

Specific Errors

```
KI_bad_precision Linear precision must be greater than zero.
                 Angular precision does not lie within allowed range
```

Description The value of 'pnum' is used to indicate which parameter to set and thereby which value ('ival' or 'rval') to use. At present this routine receives only real parameters.

The valid values of 'pnum' and the corresponding parameters are:

`pnum'	receive	parameter
SLMPLP	`rval'	Absolute (linear) precision of modeler as a real value greater than zero. Lengths and distances that differ by no more than this value will be treated as equal.
SLMPAP	`rval'	Angular precision of modeler as a real value greater than zero. Angles, and values calculated from normalized vectors, that differ by no more than this value will be treated as equal.

Modeler precision should not be changed from default values except where this is essential for compatibility with old transmit files.

SESTPA - Set state of part

Receives

```
KI_tag_part      *part    part to change
KI_cod_enst      *state   desired new state of the part
```

Returns

```
KI_cod_error     *ifail   failure indicator
```

Specific Errors

```
KI_bad_state_combn    bad combination of old and new states
```

Description SESTPA changes the state of the given part.

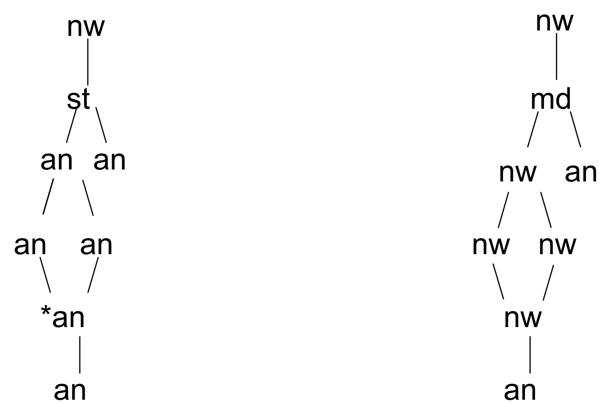
Valid combinations of the part state and the desired state are given by the following table:

		Desired Part State (ENST..)				
		ST	**AN**	**MD**	**NW**	**UN**
Given Part State	**ST**			Y	Y-	
	AN				Y	
	MD				Y-	
	NW					
	UN					

Where Y indicates a valid combination. All other combinations will return KI_bad_state_combn in ifail.

For those combinations marked with a hyphen (i.e. stored or modified to new) the key of the part is discarded.

If the original state of the part is stored or anonymous any stored or anonymous parts which instance it are also 'changed' to modified or new respectively, and any stored or anonymous parts which instance them and so on up the parts graph until a new or modified part is met. For example, the graph on the right shows the result of changing the part marked with a * in the graph on the left.



This sequence of actions is called a 'state change ripple' and can be represented in pseudo code as:

```
Ripple(P) =
        IF P is new or modified THEN RETURN
        IF P is anonymous or stored THEN
                SET P to new or modified
                FOR ALL parts Q which instance P
                Ripple(Q)
```

A state change ripple may be caused by any function which 'changes' a part.

SETLEN - Associates a tolerance value with a face, edge or vertex

Receives

KI_tag_entity	*entity	can be face, edge or vertex
double	*tol	new tolerance value

Returns

<KI_tag_list_edge>	*edges	list of new edges
KI_int_nitems	*nedges	number of edges
KI_cod_error	*ifail	failure indicator

Specific Errors

KI_bad_tolerance	new tolerance smaller than old; edge is too short for supplied tolerance; tolerance is less than half Parasolid tolerance
KI_tolerance_too_tight	SP-curve generator could not achieve tol
KI_failed_to_create_sp	SP_curve generator failed for unknown reason
KI_missing_geom	neither adjacent face has a surface, edge has no curve, face has no surface attached
KI_general_body	general body
KI_wrong_entity	unsuitable entity type

Description SETLEN alters the tolerance value associated with the given entity, which is allowed to be a face, edge or vertex. The tolerance is a resolution parameter, and represents the minimum distance (in 3-space) that a point and 'entity' must be apart in order to be regarded as distinct. It reflects the user's belief about the accuracy of the geometric data in the body.

The action of SETLEN in each case is as follows:

- FACE

  Associating a tolerance value with a face indicates that its surface is to be viewed as having a region of uncertainty around it whose extent is given by the tolerance. Attempting to set a tolerance on a face without a surface will result in an error. Currently face tolerances are not used internally.

- EDGE

  The action for edges is more complicated, and depends on whether the edge previously had Parasolid tolerance associated with it or not (i.e. whether Parasolid regarded the edge as "exact").

  The action for exact edges is as follows:

  The 3-space edge curve is used to create SP-curves on the adjacent surfaces, which are subsequently attached to the fins of the edge. These will be stored along with their bounding information as Trimmed Curves (type TYCUTR). The tolerance of the edge is to be viewed as a band of uncertainty within which all defining SP-curves lie. There will always be the following side-effects:

  - The original 3-space curve will be detached and deleted if possible.
  - The tolerances of all vertices on the edge will be increased if necessary so as to be greater than or at least equal to 'tol`. The point on the vertex will be re-computed.
  - All edges meeting at any vertex of the given edge which possess 3-space unbounded curves will have these converted to Trimmed Curve form, with the original curve as the basis curve. This is because the assumption that the vertex point lies on all edge curves meeting there will no longer hold true.
  - It will be impossible to make the edge "exact" again by using SETLEN. Once this routine has been used on an edge, it will continue to be represented geometrically by means of SP-curves attached to the fins unless MENDEN or RETLEN is called.

Additionally, the following may occur:

- It may not always be possible to compute one SP-curve from the original curve and the relevant face surface. This will be due to G1 discontinuities (in the surface parameter space) which will have to be represented as extra vertices. The edge may therefore be split a number of times, once for every surface singularity or discontinuity which lay on the original curve. Any new edges created by this process will be returned to the user in 'edges'.

It is an error to set a tolerance on an "exact" edge when the 3-space curve is not present.

It is an error if the edge is shorter than twice the given 'tol'.

The action for edges already possessing a tolerance is as follows:

The SP-curve(s) already present on the fins of the edge will be unchanged by SETLEN. The tolerance of the edge will be set to 'tol'. The tolerances of vertices of the edge will be increased if necessary so as to be greater than or at least equal to the supplied tolerance. It should be noted that reducing the tolerance value on an already toleranced edge is not recommended, as the existing SP-curve(s) will not be replaced in this case. Hence such an edge may fail CHCKEN.

- VERTEX

The tolerance of a vertex is to be viewed as the radius of a sphere of uncertainty around the point of the vertex. Setting the tolerance value is subject to the following restrictions:

- it is not permitted to set the tolerance of a vertex to less than the maximum of the tolerances of the edges meeting at it.
- attempting to set a tolerance on a vertex without a point attached will result in an error.

Note: Setting tolerances on faces, edges and vertices risks rendering the model invalid, as CHCKEN will use these tolerance values for determining whether geometric items meet appropriately or not. For example, increasing the tolerance of a particular vertex may result in its clashing with a nearby edge, or being within tolerance of another vertex. Both of these configurations will be recognized by CHCKEN. Some checks are made for local effects of this nature but such checking is by no means exhaustive.

This function is not supported for entities which are part of a general body.

SEUFEN - Set user field of entity

Receives

```
KI_tag         *tag       tag whose user field is to be set
KI_int_ufdval   ufdval[]  user-field value: array length is the
                          user-field size set by STAMOD
```

Returns

```
KI_cod_error   *ifail     error code
```

Specific Errors

```
KI_not_a_tag            'tag' is not valid
KI_no_user_fields       user fields are not available
```

Description The user field of 'tag' is set to the value given in array 'ufdval'.

SHAREN - Shares underlying geometry of a body

Receives

```
KI_tag_body      *body         body on which to attempt sharing
<KI_int_nitems>  *nopts        number of options in opts
KI_cod_shop       opts[nopts]  sharing control options
```

Returns

```
<KI_int_nitems>  *ngeom        number of geometries removed
KI_cod_error     *ifail        failure code
```

Description This function attempts to reduce the size of a part. It is primarily intended for use on pre-V5 bodies.

With no options supplied, the function will search through all the surfaces and curves in the body removing any duplicates it finds.

Duplicates are curves or surfaces of the same type that are spatially and parametrically coincident. The function is designed only to spot where copies have been made (either internally or by the user).

The option accepted is:

SHOPIC: only process intersection curves

The option SHOPIC limits the search to edge curves which are intersection curves. In pre-V5 bodies a high proportion of the cases where sharing is possible occur with these edge curves.

SIMPEN - Simplifies the underlying geometry of a body

Receives

```
KI_tag_body              *body   body to simplify
KI_cod_slle              *level  level of simplification
```

Returns

```
<KI_tag_list_entity>     *geom   list of new geometric entities
<KI_int_nitems>          *ngeom  number of new entities
KI_cod_error             *ifail  failure indicator
```

Description This function takes a body and attempts to simplify its free form geometry. The body itself, and topological nodes in the body are not affected; the geometric ones may be replaced by simpler geometries, and the old geometries deleted. Tags of all new geometric entities are returned in 'geom'.

Two levels of simplification are available:

- SLLEGL - Global simplification

  A B-curve or B-surface will only be replaced if a single curve or surface can replace the entire original.

- SLLELO - Local simplification

A B-curve or B-surface may be partially replaced by simpler curves or surfaces. A side effect of this is that edges or faces that were originally on the same B-curve or B-surface, may be on different curves or surfaces after the simplification

The following simplifications may be made on curves:

Original Form	Simplified Form
B-curve	line or circle
B-curve	B-curve with weights removed

The following simplifications may be made on surfaces

Original Form	Simplified Form
B-surface	plane, cylinder, cone, sphere, torus
B-surface	B-surface with weights removed

SPLTEN - Split topology and geometry of body at any G1 discontinuities

Receives

```
KI_tag_body       *body      body
```

Returns

```
<KI_int_nitems>    *nfaces    number of faces that have been split
<KI_tag_list_face> *faces     list of original faces that were split
<KI_tag_list_list> *new_faces list for each new set of faces,
                              corresponding to each of the old faces
KI_cod_error       *ifail     failure indicator
```

Specific Errors

```
KI_cant_extract_geom           failed to extract geom
```

Description Receives a body 'body', that maybe invalid due to geometry allocated to topology that is not G1 continuous. The faces and edges will be split at places where the continuity is G0. Existing geometry will be modified, new topology will be created - faces, edges and vertices.

A curve that is not G1, and lies on a surface that is G1 will be split, and the surface left intact.

'faces' is a list of the original faces on the body that have been split.

new_faces' is a list of face lists. Each of the face lists corresponds to one of the original faces. For example if a face has discontinuities, it will be split and new faces created. The 'face' list will contain the tag of the original face, and the element in the 'new_faces' list will contain a list of the new face's corresponding tags.

SRCHIL - Search for value in a list of ints from a starting index

Receives

```
KI_tag_list_int  *list   list which is to be searched
int              *value  value to be looked for
KI_int_index     *start  starting position to look for integer
```

Returns

```
<KI_int_index>  *index index of first item in list which matches
                       given value
KI_cod_error    *ifail failure indicator
```

Specific Errors

```
KI_bad_index    'start' not less than list length + 1
```

Description The list is searched from a starting position for a match with the given integer. The starting index cannot be less than 1, the start of the list, or greater than the list length. If no match is found 'index' is set to zero.

Can be called from the GO.

SRCHRL - Search for a value in a list of reals from a starting index

Receives

```
KI_tag_list_dbl *list  list which is to be searched
double          *value value to be looked for
KI_int_index    *start starting position to look for real
```

Returns

```
<KI_int_index>  *index index of first item in list which matches
                       given value
KI_cod_error    *ifail failure indicator
```

Specific Errors

```
KI_bad_index    'start' not less than list length + 1
```

Description The list is searched from a starting position for a match with the given real. The starting index cannot be less than 1, the start of the list, or greater than the list length. If no match is found 'index' is set to zero.

Can be called from the GO.

SRCHTG - Search for a value in a list of tags from a starting index

Receives

```
KI_tag_list_tag *list  list which is to be searched
KI_tag          *value entity to be looked for
KI_int_index    *start starting position to look for tag
```

Returns

```
<KI_int_index>  *index index of item in list which matches given value
KI_cod_error    *ifail failure indicator
```

Specific Errors

KI_bad_index	'start' not less than list length + 1

Description The list is searched from a starting position for a match with the given tag. The starting index cannot be less than 1, the start of the list, or greater than the list length. If no match is found 'index' is set to zero.

Can be called from the GO.

STAMOD - Starts the modeller

Receives

KI_cod_logical	*kijon	flags whether to write journal file or not
<KI_int_nchars>	*nchars	number of characters in journal file name (used only if kijon is set to true)
KI_chr_filename	jfilnm[nchars]	name of journal file
int	*usrfld	size of user fields (in integers)

Returns

KI_tag_entity	*world	world
int	*kivrsn	version number of interface being invoked
KI_cod_error	*ifail	failure indicator

Specific Errors

KI_cant_open_jrnl	Journal file could not be opened
KI_bad_filename	Character string is not a valid file name
KI_bad_user_field_size	User field length is negative or longer than maximum
KI_incorrect_mc_conf	Machine configuration not authorised for Parasolid
KI_modeller_not_stopped	STAMOD already called since the last call to STOMOD

Description This routine starts up the modeler and must be called before any other. It may not then be called again until STOMOD has been called. If the journal flag 'kijon' is set to true then journal records will be sent to the specified file for each call to the interface.

The length (in integers) of user fields may also be specified. The maximum length is 16 integers; specifying a length of zero will result in no user fields being allocated. Note that in order to receive an archived part with its user fields, it is necessary that the user field length for the current session be the same as when the part was archived.

The world entity (which contains all loaded assemblies and bodies) is returned.

The interface parameters are initialized: see SEINTP, for documentation of the default values.

The version number of the kernel is returned in 'kivrsn'; when read as a decimal integer it is of form Mmmbbb, where M is the major version number of Parasolid, mm is the minor version number and bbb is the build number, which identifies when the copy of Parasolid currently being run was compiled. The entire version number should always be quoted in any correspondence with your supplier of Parasolid.

On platforms where it is possible to vary the machine configuration in ways that would affect Parasolid (eg, the format of floating point numbers or the endian-ness of numbers) the modeller will not be started if the current configuration is not recognised as being safe. In this case the ifail KI_incorrect_mc_conf will be returned.

STOMOD - Stops modeller

Returns

KI_cod_error	*ifail	failure indicator

Specific Errors

KI_modeller_not_started	STOMOD already called since the last call to STAMOD

Description The modeller is closed down. No further KI routines should be called until STAMOD has been called again.

SUBBYS - Subtract bodies

Receives

KI_tag_body	*targby	body to be modified
KI_tag_list_body	*tolbys	body or list of bodies used to modify 'targby'

Returns

KI_tag_assembly	*assemb	assembly of resulting bodies
<KI_int_nitems>	*nbodys	number of bodies in 'assemb'
KI_cod_error	*ifail	failure code

Specific Errors

KI_invalid_bodies	Boolean failure or invalid bodies
KI_partial_coi_found	Boolean failure due to partial coincidence
KI_wire_body	Target or tool is a wire body
KI_missing_geom	Target or tool has incomplete geometry
KI_non_manifold	Non-manifold result
KI_same_tool_and_target	Tool body is also target body
KI_mixed_sheets_solids	Mixture of sheet and solid tool bodies
KI_instanced_tools	Instanced tool bodies
KI_duplicate_tools	Duplication in list of tool bodies
KI_not_in_same_partition	Target and tool are not all in the same partition
KI_general_body	General body

Description The target body (workpiece) is modified by removal of all regions where it overlaps one of the tool bodies (modifiers). The resulting body or bodies replace the workpiece in the world and are instanced in the new assembly returned.

The modifiers are deleted and the tags in the list 'tolbys' become dead.

If 'targby' was instanced in any assemblies, the instances are now of 'assemb'.

Boolean operations such as subtraction cannot be performed unless the bodies are completely specified geometrically.

This function does not support general bodies.

SWEENT - Sweep entity

Receives

```
KI_tag_list_topology *swept  body, vertex, face or list of faces to
                             sweep
KI_vec_displacement   path   translation vector
```

Returns

```
KI_tag_list_topology *latrls lateral edge or face(s)
KI_tag_list_topology *extent extruded vertex or edge(s)
KI_int_nitems        *nlatrl number of laterals
<KI_cod_rtlo>        *state  state of body after sweep
                              RTLOOK => Valid
                              RTLONG => Negated
                              RTLOSX => Self-intersecting
KI_cod_error         *ifail  failure code
```

Specific Errors

```
KI_su_self_intersect Sweep would produce a self-intersecting surface
KI_impossible_sweep  Cannot determine swept geometry
KI_non_manifold      Cannot sweep body with non-manifold boundary
KI_unsuitable_entity Body unsuitable for sweep
KI_wrong_entity      Sweep entity is not a body, vertex or face
```

Description The entity to be swept is moved along the path vector leaving lateral entities in its wake. Entities which may be swept, the laterals they produce and the type of the resulting body are identified in the following table.

Swept Entities	Laterals Returned	Result
Minimal body	One edge	Wire body
End vertex of wire body	One edge	Wire body
Wire body	One or more faces	Sheet body
Sheet body	One or more faces	Solid body
One or more faces of a solid body	One or more faces	Solid body
General body	One or more faces and edges	General body

For every entry in 'latrls', there is a corresponding entry in 'extent'. This list contains the topological entities extruded to create the lateral entities. For every lateral face there is a swept edge from the original body and for every lateral edge there is a swept vertex.

Suitable Sweep Entities:

- Any minimal body.
- End vertices (in contradistinction to interior vertices) of wires.
- Any wire body, either an open or closed loop of edges.
- Any sheet body whose boundary is manifold.
- Any group of manifold faces of a solid body. Entire shells must not be contained within the list.
- Any general body with only one region and no edges with more than two faces.

Attempting to sweep an entity which is not one of the above will give rise to the ifail KI_unsuitable_entity.

An entity which has an edge whose attached curve is of the type TYCUIN (i.e. intersection curve), TYCUFG (i.e. foreign geometry), TYCUSP (i.e. sp-curve) or type TYCUCP (i.e. constant parameter curve) cannot be swept. However tolerant edges and sheet bodies, which have curves of type TYCUSP (ie sp-curves) attached to their fins, can be swept.

After a sweep any coincident topology will not be fused or united.

In general this procedure may give rise to self-intersecting body boundaries, which could cause unpredictable errors later.

Any new surfaces created are analytic surfaces if possible. If it is not possible to use an analytic surface then the type of the new surface will depend on SLIPGS (see SEINTP and OUINTP). If the creation of generated surfaces is off, then any new non-analytic surfaces will be B-surfaces; if it is on, then they will be swept surfaces.

If local checking is on, (see SEINTP and OUINTP) consistency checks will be made on newly created topological and geometrical entities, and the state of the body returned. A state of RTLOOK indicates the body is valid. A state of RTLONG indicates that the result body was originally "inside out" but has been negated, and is now "positive" (has positive volume) and valid. A state of RTLOSX indicates the body is self-intersecting and further modelling operations on it may fail.

If the session parameter for local checking is switched off, the state returned will be RTLOOK regardless.

A self-intersecting body can be returned even if the ifail is zero.

For general bodies, the body is copied, the copy is transformed, corresponding vertices on the body and its copy are joined with lateral faces. All the lateral entities are returned, both edges and faces. The state of RTLONG is never returned since this has no meaning for general bodies.

SWIENT - Swing entity

Receives

```
KI_tag_list_topology *swung  body, vertex, face or list of faces
                             to swing
KI_vec_position       point  point on axis of rotation
KI_vec_axis           direct direction of axis of rotation
KI_dbl_angle         *angle  angle of swing (in radians)
```

Returns

```
KI_tag_list_topology *latrls lateral edge or face(s)
KI_tag_list_topology *extent extruded vertex or edge(s)
KI_int_nitems        *nlatrl number of laterals
KI_cod_rtlo          *state  state of the body after the swing
                              RTLOOK => Valid
                              RTLONG => Negated
                              RTLOSX => Self-intersecting
KI_cod_error         *ifail  failure code
```

Specific Errors

KI_su_self_intersect	Swing would produce a self intersecting surface
KI_impossible_swing	Cannot determine swung geometry
KI_non_manifold	Cannot swing body with non-manifold boundary
KI_bad_angle	Angle must be in the range -2pi <= angle <= 2pi; Full swing impossible in this context
KI_unsuitable_entity	Body unsuitable for swing
KI_wrong_entity	Swing entity not a body, vertex or face

Description The entity to be swung is moved along an arc specified by the axis leaving lateral entities in its wake. Entities which may be swung, the laterals they produce and the type of the resulting body are identified in the following table.

Swept Entities	Laterals Returned	Result
Minimal body	One edge	Wire body
End vertex of wire body	One edge	Wire body
Wire body	One or more faces	Sheet body
Sheet body	One or more faces	Solid body
One or more faces of a solid body	One or more faces	Solid body
General body	One or more faces and edges	General body

For every entry in 'latrls', there is a corresponding entry in 'extent'. This list contains the topological entities extruded to create the lateral entities For every lateral face there is a swung edge from the original body and for every lateral edge there is a swung vertex.

Suitable Swing Entities:

- Any minimal body not coincident with the axis of rotation.
- End vertices (in contradistinction to interior vertices) of wires not coincident with the axis of rotation.
- Any wire body, either an open or closed loop of edges.
- Any sheet body whose boundary is manifold.
- Any group of manifold faces of a solid body. Entire shells must not be contained within the list.
- Any general body with only one region and no edges with more than two faces.

Attempting to swing an entity which is not one of the above will give rise to one of the ifails KI_wrong_entity, KI_unsuitable_entity or KI_non_manifold as appropriate.

Permissible Swings:

- An entity may not be swung through more than 2pi or less than -2pi. It is not possible to perform full swings (where the angle is 2pi or -2pi) on faces of a solid or end vertices of a wire.
- For a wire body none of the edges may be coincident with the axis, nor are they allowed to intersect the axis at any points other than at the ends of the wire.
- For full swings of a sheet body the axis may not intersect the sheet at a single point, but it may be coincident with any edges of the sheet.
- For partial swings of a sheet body or faces from a solid body the restriction is that the axis may not intersect with any edges tangentially. One special case is allowed when the curve of the edge is a circle orthogonal and tangential to the swing axis.
- Bodies of revolution are created using a full swing. The spun entities (the vertex of a minimal body, the edges of a wire, or the faces of a sheet) are not deleted.

An entity which has an edge whose attached curve is of the type TYCUIN (i.e. intersection curve), TYCUFG (i.e. foreign geometry), TYCUSP (i.e. sp-curve), or TYCUCP (i.e. constant parameter curve) cannot be swung. However tolerant edges and sheet bodies, which have curves of type TYCUSP (i.e. sp-curves) attached to their fins, can be swung.

After a swing any coincident topology will not be fused or united.

In general this procedure may give rise to self-intersecting body boundaries, which could cause unpredictable errors later.

Any new surfaces created are analytic surfaces if possible. If it is not possible to use an analytic surface then the type of the new surface will depend on SLIPGS (see SEINTP and OUINTP). If the creation of generated surfaces is off, then any new non-analytic surfaces will be B-surfaces; if it is on, then they will be swung surfaces.

If local checking is on, (see SEINTP and OUINTP) consistency checks will be made on newly created topological and geometrical entities, and the state of the body returned. A state of RTLOOK indicates the body is valid. A state of RTLONG indicates that the result body was originally "inside out" but has been negated, and is now "positive" (has positive volume) and valid. A state of RTLOSX indicates the body is self-intersecting and further modelling operations on it may fail.

If the interface parameter for local checking is switched off, the state returned will be RTLOOK regardless.

A self-intersecting body can be returned even if the ifail is zero.

For general bodies, the body is copied, the copy is transformed, corresponding vertices on the body and its copy are joined with lateral faces. All the lateral entities are returned, both edges and faces. The state of RTLONG is never returned since this has no meaning for general bodies.

TAPFAS - Taper faces in a body

Receives

```
KI_tag_list_face       *faces   face(s) to be tapered
KI_vec_position         point   point on taper plane
KI_vec_direction        direct  normal to taper plane
KI_dbl_angle           *angle   taper angle (in radians)
```

Returns

```
<KI_tag_list_face>    *flist   tapered face(s)
<KI_int_nitems>       *nfaces  number of tapered faces
KI_cod_rtlo           *state   state of the body
                                RTLOOK => Valid
                                RTLONG => Negated
                                RTLOSX => Self-Intersecting
KI_cod_error          *ifail   failure code
```

Specific Errors

```
KI_impossible_taper Taper cannot be performed
KI_cant_do_tweak    Cannot taper rubber face; Taper can only work on
                    a solid body
KI_not_in_same_part Faces must be from same part
KI_general_body     Taper does not work for general
KI_bad_angle        Angle out of range
```

Description Faces to be tapered are changed such that at all points the angle between the face normal direction and the vector 'direct' is decreased by 'angle'. If 'angle' is negative, the angle will be increased. Curves and points of the edges and vertices of the faces are recalculated. Either one face or a list of faces which all belong to the same body can be tapered.

Where edges or faces (sharing a vertex or edge at the boundary of the faces to be tapered) have geometry which is inextendable, it may not be possible to recalculate vertices or edges. This is because the curve or surface does not intersect the tapered surface. With such edges which have a user defined tolerance this is likely to occur since SP-curves are inextendable.

'Angle' is given in radians and must have an absolute value of less than pi. The arguments 'point' and 'direct' together define a planar taper surface. The cross section of the body in this surface is unchanged.

Only planar surfaces which are not parallel to the taper surface, and cylindrical and conical surfaces which have an axis parallel to 'direct' can be tapered. All other faces in the list will be ignored and will not be altered. Planar surfaces will remain planar, cylindrical surfaces will become conical and conical surfaces will be given a different angle of divergence or become cylindrical.

A list of all of the faces which were tapered and the number of entries in the list is returned. If no faces are tapered the null tag is returned.

The amount of taper must not be so much that the topology of the body would be altered, for instance if a tapered face no longer intersects an adjacent face. Faces which are tangential to another face cannot be tapered, unless that face is also tapered at the same time.

In general this procedure may give rise to self-intersecting object boundaries which could cause unpredictable errors later.

If local checking is on, (see SEINTP and OUINTP) consistency checks will be made on newly created topological and geometrical entities, and the state of the body returned. A state of RTLOOK indicates the body is valid. A state of RTLONG indicates that the result body was originally "inside out" but has been negated, and is now "positive" (has positive volume) and valid. A state of RTLOSX indicates the body is self-intersecting and further modelling operations on it may fail. It is the responsibility of the calling routine to make any necessary reparation.

If the session parameter for local checking is switched off, the state returned will be RTLOOK regardless.

A self-intersecting body can be returned even if the ifail is zero.

This function is not supported for general bodies.

THIKEN - Thickens a sheet body into a solid

Receives

```
KI_tag_body          *body    body to be thickened
<KI_dbl>             *front   thickness on front of faces
<KI_dbl>             *back    thickness on back of faces
KI_cod_logical       *check   level of checking required
KI_dbl_distance      *tol     tol of SP curve conversions
<KI_int_nitems>      *mxflt   maximum number of entities in
                     s        badtag
```

Returns

```
<KI_tag_list_entity> *oldtop original topology
<KI_tag_list_entity> *newtop new topology
<KI_tag_list_entity> *badtag entities which caused problems
KI_cod_rtof          *state  state of body after offset
KI_cod_error         *ifail  failure code
```

Specific Errors

```
KI_cant_thicken      thickening failure
KI_bad_tolerance     proposed edge tolerance is too small
KI_not_sheet         body is not sheet
KI_bad_thickness     total thickness zero
```

Description The sheet 'body' is thickened by 'front' outwards (i.e. in the direction of the face normal) and 'back' inwards. Either 'front' or 'back' may be negative, but the total thickness must be non-zero.

Only manifold sheet bodies can be thickened.

Some edge geometry on the sheet boundary may have to be approximated by SP-curves in order to generate a boundary curve on the offset faces. If this is necessary, such edges will have a tolerance less than or equal to the tolerance supplied through the 'tol' argument.

Two situations can give rise to changes in the topology :

- Dealing with geometry which fails to offset. If it is known that the offset surface of a face would be self-intersecting an attemp is made to remove the face. For example, a blend may be removed from an edge. The investigation of self-intersection is not exhaustive, however, and it can occur that instances are not trapped.
- Dealing with configurations which can be repaired. For instance an edge may offset to a point or a face become absorbed into the body.

The extent to which cheking is applied to the body is specified by the 'check' argument. If 'check' is true then face-face checks are done on the body in addition to default checks. For most applications setting 'check' false will give an adequate level of checking.

Thickening which would lead to a self-intersecting or non-manifold solid is not allowed. Some of these cases will not be detected when face-face checks are not done, in this case the resulting body will be invalid.

Boundary edges with the following curves on these surfaces will be treated exactly :

Surface	**Curve**
Plane	Line, circle, ellipse or B-curve
Cylinder	Line, circle or B-curve equivalent
Cone	Line, circle or B-curve equivalent
Sphere	Circle
Torus	Circle
Swept surface or B-surface equivalent	Line, profile curve or B-curve equivalent
Spun surface or B-surface equivalent	Circle, profile curve or B-curve equivalent

Thickening only handles the cases where each vertex on the boundary of the sheet meets one of the following requirements. Either the vertex is smooth (surface normals constant on all adjoining faces) or both adjoining boundary edges must have curves of the types treated exactly (see table above) or the thickening is into the angle of the edge (i.e. convex edges inwards only and concave edges outwards only).

Each new face in the resulting body will have been generated by either a face or an exterior edge in the original sheet. The pairs of original faces or edges and corresponding new faces are returned in the 'oldtop' and 'newtop' arguments. Both lists may contain null tags where faces have vanished. If the total thickness is positive the tag of a face in the original body will remain on the front of the resulting body with the same sense otherwise it will be on the back with the opposite sense.

The error reporting scheme comprises the three arguments 'badtag', 'state' and 'ifail'. A non-zero 'ifail' is reserved for reporting unsuitable arguments to the function and system errors.

Algorithmic failures where the item causing the failure can be identified result in a zero 'ifail' and more specific information being returned in the 'state' argument and 'badtag'. For example, a thickness so large that any of the surfaces could not be offset will return 'state' RTOFSO and the tags of faces whose surfaces could not be offset in the 'badtag' list.

'state' refers to the validity of the item after modification and not to its original validity. After such a failure 'body' may be corrupt and a rollback should be performed. For this reason tags of new topology cannot be returned in 'badtag' and the tags will refer to adjacent faces in the original body supplied. For example if a new edge cannot be modified a 'state' RTOFEM is returned and 'badtag' will contain the tag of a list of the two faces whose offsets are to be used to find the new curve.

Possible values of 'state' and the contents of 'badtag' are :

Token	Tag in badtag	Meaning
RTOFOK	null	Body is OK
RTOFSO	face	Surface failed to offset or face coould not be deleted
RTOFVM	vertex list faces	Failed to find newgeometry for vertex Failed to find geometry for new vertex
RTOFEM	edge list faces	Failed to find new geometry for edge Failed to find geometry for new edge
RTOFSS	edge	Failed to find side surface
RTOFSC	vertex	Failed to find side curve
RTOFVT	edge	Edge should have disappeared
RTOFFA	face	Face failed checks
RTOFSX	list faces	Face-face inconsistency found

Notice that a successful execution of the thickening operation is indicated by :

ifail returning KI_no_errors

state returning RTOFOK

TRIMSH - Trims the sheet body to the curves

Receives

```
KI_tag_body    *sheet           sheet body to trim
KI_int_nitems  *ncurvs          number of curves supplied
KI_tag_curve    curvs[ncurvs]   curves to trim sheet
KI_int_nitems  *nopts           no of trimming options supplied
KI_cod_sltr     opts[nopts]     trimming options
KI_tag_list     optdata[nopts]  trimming option data
```

Returns

```
KI_cod_error   *ifail           failure code
```

Specific Errors

```
KI_bad_sharing            trim would give illegal sharing
KI_sheet_untrimmed        curves didn't trim sheet
KI_failed_to_trim         unable to trim sheet
KI_unsuitable_entity      sheet has more than one non-rubber face
KI_invalid_geometry       invalid trimming curve supplied
KI_fragment               trim would either fragment or delete the sheet
KI_missing_geom           sheet has no surface attached
KI_not_sheet              body is not a sheet
KI_contradictory_request  invalid combination of options
KI_bad_selection_code     inappropriate property
KI_bad_option_data        inappropriate option data
KI_duplicate_list_item    curve geometry duplicated in lists
KI_not_in_same_partition  Sheet and surf are in different partitions
```

Description This function trims a sheet body with a list of curves. It will not allow the sheet to be broken into two or more parts and will not allow the sheet to be extended or deleted.

The sheet must have only one face which must have a surface attached. If the sheet has more than one face then the error `KI_unsuitable_entity' will be returned.

Each of the trimming curves must lie on the face of the sheet. If any of the curves do not lie on the face then that curve will be ignored. If any of the curves are of type 'TYCUSP' (SP-curve) then they must reference the surface of the sheet or a copy of this surface. If this is not the case the error 'KI_invalid_geometry' will be returned.

Any curves of type TYCUTR (trimmed curve) will be deleted and their underlying curves attached to the new inscribed edges (or fins in the SP-curve case) of the sheet. It is illegal to supply the same trimmed curve more than once in 'curvs'; this will result in the error `KI_duplicate_list_item'. This is because it is impossible to share the same trimmed curve between edges (or fins).

The trimming curves will be scribed onto the the sheet creating a number of regions on the face. One of the selection options should then be used to identify which regions will survive and which will be deleted.

TRIMSH has three options: two 'SLTRKE', and 'SLTRRE' are mutally exclusive with use of one or the other compulsory, the third 'SLTRTL' is optional.

TOKEN	DATA	MEANING
SLTRKE	List of 'points' (each 3 doubles)	Keep all regions on the sheet containing at least one of these points.
SLTRRE	List of 'points' (each 3 doubles)	Remove all regions on the sheet containing at least one of these points.
SLTRTL	Single double tolerance	Attempt to remove gaps between SP-curves in the trimming set that are smaller than the sup plied tolerance.

The option 'SLTRKE' may be used to supply a list of points identifying the regions to remain on the trimmed sheet. With this option all the regions not identified by a point will be deleted.

Alternatively the option 'SLTRRE' may be used to supply a list of points identifying the regions to be deleted from the sheet. With this option all the regions not identified by a point will survive.

One of these options must be supplied although it is not possible to supply both together.

The points should be supplied in 'optdata' as a list of doubles of length 3*npoints, where npoints is the number of selection points.

If a supplied point lies on an edge or vertex of the sheet, or on one of the trimming curves, then all the regions adjacent to that point will be selected to be either kept or removed.

If any of the supplied points do not lie on the face then no regions will be selected by that point.

In some cases the trimming curves and points will be such that no region of the sheet will have been removed. In these cases TRIMSH will return the ifail 'KI_sheet_untrimmed'. An example of this would be an attempt to cut a sheet in half with a trimming curve that is too short to divide it.

Contiguous trimming curves should join to within modeller resolution. In the case of gaps larger than this, around SP-curves only, the option 'SLTRTL' is available to force TRIMSH to try to close them by modifying the SP-curve geometry. However an attempt to close gaps crossing a degeneracy or seamline of the surface parametrisation should be avoided.

'SLTRTL' has a single optional double as associated data. This double, if provided, states the maximum size of gap to be closed. Where gaps smaller than this are found, the SP-curves will be modified so that the trimming curves meet to within modeller resolution. Larger gaps will be left alone. If the optional double is not provided, then the maximum size of gap closed will default to 100 times the default resolution of the modeller.

Where the 'SLTRTL' option is not requested, TRIMSH will not make any attempt to close gaps between the trimming curves.

TRSHCU - Trim sheet body with curves

Receives

```
KI_tag_body         *sheet          sheet body to trim
KI_int_nitems       *ncurvs         number of curves supplied
KI_tag_curve         curvs[ncurvs]  curves to trim sheet
KI_int_nitems       *nopts          no of trimming options supplied
KI_cod_trsh          opts[nopts]    trimming options
KI_vec_direction     direct         direction of projection
```

Returns

```
<KI_tag_list_edge>  *edges          edges imprinted
<KI_int_nitems>     *nedges         number of edges
<KI_tag_list_curve> *which          which curve edge is derived from
<KI_tag_list_int>   *original       whether edge is original
KI_cod_error        *ifail          failure code
```

Specific Errors

```
KI_curves_dont_meet      curves not a closed loop
KI_failed_to_trim        trim would not split the sheet
KI_fragment              trim would fragment the sheet
KI_unsuitable_entity     geometry of wrong type
KI_invalid_geometry      geometry fails to pass checks
KI_bad_selection_code    inappropriate property
KI_contradictory_request invalid combination of options
KI_not_sheet             body is not a sheet
```

Description This function imprints a list of curves onto a sheet body. It will also trim the sheet body if requested. Which part of the sheet body is to be kept is controlled by options as described below. The sheet body may have any number of faces. The method of projection of the curves is also controlled by the options. Consecutive curves in a closed loop or open string of curves should meet at their ends to modeller tolerance.

The returned arguments are 3 parallel lists - the first of these is a list of the edges imprinted; the second is a list of curve tags which is used to indicate which curve each edge is derived from and the third is a list of logicals indicating whether each edge is an original edge (where a curve has projected onto a existing edge) or a new one. KI_true means that the edge is original.

The only method of projection supported at present is where the curves are projected onto the body along a given direction given by the 'direct' argument. the option for this is TRSHPD. Each point on one of the curves should only have one image under projection on the body - i.e. the sheet should not be self-obscuring when viewed down the projection direction.

The option TRSHTR controls whether the function should also perform a trim operation on the sheet. If this is not supplied then no trimming is done and the operation is similar to a SCRIBE or IMPRNT. If it supplied then a further token must be supplied which indicates which face-set of the sheet is to survive the trimming operation:

Token	Meaning
TRSHLC	The face-set to the left of the first curve will be retained.
TRSHRC	The face-set to the right of the first curve will be retained.
TRSHIL	The face-set inside the loop of curves will be retained.
TRSHOL	The face-set outside the loop of curves will be retained.

These last two options must only be supplied if the curves provided constitute a closed, planar loop. If TRSHTR is selected then the imprinted edges must split the sheet into at least two face-sets.

The curves supplied must be of type TYCUPA, TYCUEL, TYCUCI or TYCUTR. If the y are trimmed curves then the underlying curves must be of type TYCUPA, TYCUEL, TYCUCI or TYCUST.

TWEFAC - Transform geometry of faces

Receives

```
KI_tag_list_entity    *faces  face(s) to be transformed
KI_tag_list_transform *transf list of transforms (move and/or rotate
                              only)
```

Returns

```
KI_cod_rtlo            *state state of the body
                               RTLOOK => Valid
                               RTLONG => Negated
                               RTLOSX => Self-Intersecting
KI_cod_error          *ifail  failure code
```

Specific Errors

```
KI_cant_do_tweak        Tweak cannot be performed; Transform cannot be
                        applied to a rubber face
KI_general_body         General bodies not supported
KI_list_wrong_length    Incorrect number of transforms for faces
KI_duplicate_list_item  Face appears twice in argument list
KI_not_in_same_part     All faces are not from the same part
KI_wrong_transf         Transform is identity, or a reflection
KI_wrong_entity         Faces is not of expected type
```

Description The surfaces of the faces are modified by the transformations and the curves and points of their edges and vertices are recalculated.

Where edges or faces (sharing a vertex or edge at the boundary of the faces to be transformed) have geometry which is inextendable, it may not be possible to recalculate vertices or edges. This is because the curve or surface does not intersect the transformed surface. With such edges which have a user defined tolerance this is likely to occur since SP-curves are inextendable.

In general this procedure may give rise to self-intersecting object boundaries which could cause unpredictable errors later.

If local checking is on, (see SEINTP and OUINTP) consistency checks will be made on newly created topological and geometrical entities, and the state of the body returned. A state of RTLOOK indicates the body is valid. A state of RTLONG indicates that the result body was originally "inside out" but has been negated, and is now "positive" (has positive volume) and valid. A state of RTLOSX indicates the body is self-intersecting and further modelling operations on it may fail. It is the responsibility of the calling routine to make any necessary reparation.

If the interface parameter for local checking is switched off, the state returned will be RTLOOK regardless.

A self-intersecting body can be returned even if the ifail is zero.

The arguments to the routine allow the user to perform the transformation in a number of ways. These are as follows :-

- one face, one transform

  e.g. f1 -> tr1
- list of faces, one transform

  e.g. (f1 f2 f3) -> tr1
- list of faces, list of transforms, the length of the list of transforms being either equal to the length of the list of faces, or being a list of length one

  e.g. (f1 f2 f3) -> (tr1 tr2 tr3) or (f1 f2 f3) -> (tr1)
- list of lists of faces, one transform

  e.g. ((f1 f2 f3) (f4 f5)) -> tr1
- list of lists of faces, list of transforms, the length of the list of transforms being either equal to the length of the list of the lists of faces, or being a list of length one

  e.g. ((f1 f2 f3) (f4 f5)) -> (tr1 tr2) or ((f1 f2 f3) (f4 f5)) -> (tr1)

This is not supported for general bodies.

TWSUFA - Tweak the surface(s) of face(s)

Receives

```
KI_tag_list_face       *faces    face(s) to tweak
KI_tag_list_surface    *surfs    new surface(s) for face(s)
KI_tag_list_int        *senses   senses for surfaces
```

Returns

```
KI_cod_rtlo          *state   state of the body
                               RTLOOK => Valid
                               RTLONG => Negated
                               RTLOSX => Self-Intersecting
KI_cod_error         *ifail   failure code
```

Specific Errors

```
KI_cant_do_tweak       Tweak cannot be performed; Tweak cannot be
                       applied to face on boundary; Tweak cannot be
                       applied to a rubber face
KI_bad_sharing         attempt to illegally share a surface
KI_general_body        Attempt to illegally share a surface
KI_invalid_geometry    Surface fails checks
KI_unsuitable_entity   Surface is too complex
KI_not_same_length     Lists not of equal length
KI_duplicate_list_item Face appears twice in argument list
KI_not_in_same_part    All faces are not from the same part
```

Description The surface geometries of the faces are replaced and the curves and points of edges and vertices are recalculated.

Where edges or faces (sharing a vertex or edge at the boundary of the faces to be tweaked) have geometry which is inextendable, it may not be possible to recalculate vertices or edges. This is because the curve or surface does not intersect the new surface. With such edges which have a user defined tolerance this is likely to occur since SP-curves are inextendable.

If the faces are on a sheet body, they must not share an edge with any rubber faces.

The entries in 'senses' define whether the corresponding surface is to be reversed before being attached to a face. After the tweak operation has been performed all the entities in 'faces' will have their senses set to true.

The number of surfaces and the number of senses must be identical. The number of surfaces supplied must either be one, in which case every face is tweaked to the same surface, or equal to the number of faces, in which case every face is tweaked to the corresponding surface.

The self intersection check is only performed if the appropriate option is set (see SEINTP).

In general this procedure may give rise to self-intersecting object boundaries which could cause unpredictable errors later.

If local checking is on, (see SEINTP and OUINTP) consistency checks will be made on newly created topological and geometrical entities, and the state of the body returned. A state of RTLOOK indicates the body is valid. A state of RTLONG indicates that the result body was originally "inside out" but has been negated, and is now "positive" (has positive volume) and valid. A state of RTLOSX indicates the body is self-intersecting and further modelling operations on it may fail. It is the responsibility of the calling routine to make any necessary reparation.

If the interface parameter for local checking is switched off, the state returned will be RTLOOK regardless.

A self-intersecting body can be returned even if the ifail is zero.

This function is not supported for general bodies.

UNIBYS - Unite bodies

Receives

```
KI_tag_body      *targby target body
KI_tag_list_body *tolbys body or list of bodies used to modify 'targby'
```

Returns

```
KI_tag_assembly  *assemb assembly of resulting bodies
KI_int_nitems    *nbodys number of bodies in 'assemb'
KI_cod_error     *ifail  failure code
```

Specific Errors

KI_invalid_bodies	Boolean failure or invalid bodies
KI_partial_coi_found	Boolean failure due to partial coincidence
KI_t_sheet	T-sheet
KI_opposed_sheets	Attempt to unite opposed sheets
KI_wire_body	Target or tool is a wire body
KI_missing_geom	Target or tool has incomplete geometry
KI_non_manifold	Non-manifold result
KI_same_tool_and_target	Tool body is also target body
KI_cant_unite_solid_sheet	Attempt to unite solid and sheet
KI_mixed_sheets_solids	Mixture of sheet and solid tool bodies
KI_instanced_tools	Instanced tool bodies
KI_duplicate_tools	Duplication in list of tool bodies
KI_general_body	General body
KI_not_in_same_partition	Target and tools are not all in the same partition

Description The target body (workpiece) is extended by inclusion of all regions of space contained in one of the tool bodies (modifiers). The resulting body or bodies replace the workpiece in the world and are instanced in the new assembly returned.

The modifiers are deleted and the tags in the list 'tolbys' become dead.

If 'targby' was instanced in any assemblies, the instances are now of 'assemb'.

Boolean operations such as union cannot be performed unless the bodies are completely specified geometrically.

This function does not support general bodies.

UNLDPA - Unload part

Receives

```
KI_tag_part          *part        part to unload
```

Returns

```
KI_cod_error         *ifail       failure indicator
```

Specific Errors

```
KI_part_not_keyed   part is not keyed and cannot be unloaded
```

Description UNLDPA removes 'part' from internal memory. The part must be a stored part otherwise KI_part_not_keyed is returned in 'ifail'.

We define the true-sub-parts of some stored (ENSTST) part S as those anonymous (ENSTAN) sub-parts of S reachable from S without encountering other stored parts.

The contents of the part, including all anonymous true-sub-parts are deleted from internal memory and removed from the world. All associated data of the part is also deleted. The part itself remains attached to the world but its state is changed to unloaded (ENSTUN).

The part may be instanced.

The system is able to determine the key and box of an unloaded part.

If a later operation requires access to the contents of an unloaded part it will be reloaded automatically.

Kernel Interface Tokens A

A.1 Introduction

This appendix lists all the tokens used by the kernel interface, grouped by usage.

A.2 List of Kernel Interface Tokens Grouped by Usage

Null Tag Value

NULTAG	0	value for a null tag

Tokens KI_false KI_true

KI_false	0	KI_cod_logical value for false
KI_true	1	KI_cod_logical value for true

Interface Param SLIP-- (0+)

SLIP00	0	interface param <base>
SLIPCH	1	checking
SLIPJO	2	journalling
SLIPBB	3	bulletin board
SLIPRB	5	rollback
SLIPBT	6	binary transmit/receive
SLIPLC	7	local checking
SLIPUF	8	receive user-fields
SLIPSN	9	binary snapshot
SLIPSI	10	self intersecting checking
SLIPCO	12	continuity checking
SLIPDC	14	data checking
SLIPTL	15	tag limit
SLIPGS	16	create generated surfaces
SLIPRF	17	roll forward
SLIPGT	18	create generalised topology

Modeling Param SLMP-- (0+)

SLMP00	0	modeling param <base>
SLMPLP	1	linear precision
SLMPAP	2	angular precision

Reason For Abort SLAB-- (0+)

SLAB00	0	reason for abort <base>
SLABUI	1	user interrupt
SLABRE	2	run-time error
SLABFE	3	Frustrum error

Rendering Option RROP-- (0+)

RROP00	0	rendering option <base>
RROPED	1	render edges
RROPRH	2	render radial hatch
RROPPH	3	render planar hatch
RROPUB	4	render unfixed blends
RROPSI	5	render silhouettes
RROPIV	6	render invisible lines
RROPTR	7	render transformed entities
RROPPS	8	perspective view
RROPSM	9	distinguish smooth edges
RROPSF	11	surface reflectivity
RROPBK	12	background
RROPFC	13	face color
RROPAN	15	anti-alias
RROPDM	18	depth modulation
RROPRG	19	regions
RROPRA	20	regions by attribute
RROPIS	21	distinguish image smoothness
RROPPA	22	render parametric hatch
RROPTL	23	render translucent faces
RROPCT	24	curve tolerances
RROPCV	25	polygon convexity
RROPFS	26	facet size
RROPHO	27	holes permitted
RROPNF	28	no fitting
RROPPT	29	planarity tolerances
RROPST	30	surface tolerances

RROPVN	31	output vertex normals
RROPET	32	output edge tags
RROPIE	33	render internal edges
RROPIN	34	distinguish internal edges
RROPPC	35	render parametric curves
RROPVM	36	match facet vertices
RROPDR	37	render drafting lines
RROPHR	38	hierarchical output
RROPHN	39	hierarchical output no geometry
RROPFI	40	facet with infinite view dependency
RROPFP	41	facet with perspective view dependency
RROPTS	42	facet strips
RROPNC	43	render nurbs curve
RROPMF	44	minimum facet size
RROPPI	45	parameter information
RROPDS	46	drafting / smooth edges behaviour
RROPHP	47	hierarchical output parametrised
RROPVP	48	use viewport
RROPAS	49	analytic silhouettes
RROPD1	50	first derivatives at facet vertices
RROPD2	51	first and second derivatives at facet vertices
RROPIL	52	ignore loops
RROPSD	53	silhouette density

Entity Type TYEN-- (1000+)

TYEN00	1000	entity type <base>
TYENGE	1001	geometry entity
TYENTO	1002	topology entity
TYENAD	1003	assoc data entity

Geometry Type TYGE-- (2000+)

TYGE00	2000	geometry type <base>
TYGEPT	2001	point
TYGECU	2002	curve
TYGESU	2003	surface
TYGETF	2004	transform

Point Type TYPT-- (2500+)

TYPT00	2500	point type <base>
TYPTCA	2501	cartesian point

Curve Type TYCU-- (3000+)

TYCU00	3000	curve type <base>
TYCUST	3001	straight
TYCUCI	3002	circle
TYCUEL	3003	ellipse
TYCUIN	3004	intersection-curve
TYCUPA	3005	B-curve
TYCUSP	3006	sp-curve
TYCUFG	3007	foreign curve
TYCUCP	3008	constant parameter curve
TYCUTR	3009	trimmed curve

Surface Type TYSU-- (4000+)

TYSU00	4000	surface type <base>
TYSUPL	4001	plane
TYSUCY	4002	cylinder
TYSUCO	4003	cone
TYSUSP	4004	sphere
TYSUTO	4005	torus
TYSUPA	4006	parametric-surface
TYSUBL	4007	blending-surface
TYSUOF	4008	offset-surface
TYSUSE	4009	swept-surface
TYSUSU	4010	swung-surface
TYSUFG	4011	foreign surface

Blending Sub Types TYBL-- (4600+)

TYBL00	4600	blending sub types <base>
TYBL1B	4601	non_overlapped
TYBL2S	4602	2 overlapping, same sense
TYBL2D	4603	2 overlapping, different sense
TYBL3S	4604	3 overlapping, same sense
TYBL3D	4605	3 overlapping, different sense

Topology Type TYTO-- (5000+)

TYTO00	5000	topology type <base>
TYTOVX	5001	vertex
TYTOED	5002	edge
TYTOLO	5003	loop
TYTOFA	5004	face
TYTOSH	5005	shell
TYTOBY	5006	body
TYTOIN	5007	instance
TYTOAS	5008	assembly
TYTOWO	5009	world
TYTOFN	5010	fin
TYTORG	5011	region

Assembly Type TYAS-- (5050+)

TYAS00	5050	assembly type <base>
TYASCL	5051	collective assembly

Instance Type TYIN-- (5070+)

TYIN00	5070	instance type <base>
TYINPS	5071	positive instance

Vertex Property ENVE-- (5100+)

ENVE00	5100	vertex property <base>
ENVEIS	5101	isolated vertex
ENVESP	5102	spur vertex
ENVEWR	5103	wire vertex
ENVENO	5104	normal vertex

Edge Property ENED-- (5300+)

ENED00	5300	edge property <base>
ENEDOW	5301	open wire edge
ENEDON	5302	open normal edge
ENEDCN	5303	closed normal edge
ENEDCW	5304	closed wire edge
ENEDRN	5305	ring normal edge
ENEDOB	5306	open biwire edge

ENEDCB	5307	closed biwire edge
ENEDRB	5308	ring biwire edge

Loop Property ENLO-- (5400+)

ENLO00	5400	loop property <base>
ENLOHO	5401	hole loop
ENLOPE	5402	peripheral loop
ENLONA	5403	loop not hole or peripheral

Shell Property ENSH-- (5500+)

ENSH00	5500	shell property <base>
ENSHSO	5501	solid shell
ENSHVO	5502	void shell
ENSHSH	5503	sheet shell
ENSHWR	5504	wire shell

Body Property ENBY-- (5600+)

ENBY00	5600	body property <base>
ENBYSO	5601	solid body
ENBYSH	5602	sheet body
ENBYMN	5603	minimum body
ENBYWR	5604	wire body
ENBYRG	5605	general body

Wire Property ENWR-- (5620+)

ENWR00	5620	wire property <base>
ENWRGN	5621	general wire
ENWRPA	5622	parametric wire

Sheet Property ENSE-- (5640+)

ENSE00	5640	sheet property <base>
ENSEGN	5641	general sheet
ENSEPA	5642	parametric sheet

Part State ENST-- (5660+)

ENST00	5660	part state <base>
ENSTST	5661	stored part
ENSTMD	5662	modified part

ENSTNW	5663	new part
ENSTAN	5664	anonymous part
ENSTUN	5665	unloaded part

Enclosure ENCL-- (5700+)

ENCL00	5700	enclosure <base>
ENCLIN	5701	inside
ENCLOU	5702	outside
ENCLON	5703	on (the limits of)

Attribute Class RQAC-- (5800+)

RQAC00	5800	attribute class <base>
RQAC01	5801	attribute class 1
RQAC02	5802	attribute class 2
RQAC03	5803	attribute class 3
RQAC04	5804	attribute class 4
RQAC05	5805	attribute class 5
RQAC06	5806	attribute class 6
RQAC07	5807	attribute class 7

Attribute Property RQAP-- (5900+)

RQAP00	5900	attribute property <base>
RQAPIN	5901	integer property
RQAPRL	5902	real property
RQAPCS	5903	character property
RQAPVC	5904	vector property
RQAPCO	5905	coordinate property
RQAPDR	5906	direction property
RQAPAX	5907	axis property

Assoc Data Type TYAD-- (6000+)

TYAD00	6000	assoc data type <base>
TYADAT	6001	attribute
TYADLI	6002	list
TYADAD	6003	attribute definition
TYADFE	6005	feature

Attribute Type TYAT-- (7000+)

TYAT00	7000	attribute type <base>
TYATSY	7001	system attribute
TYATUS	7002	user attribute

System Attribute Type TYSA-- (8000+)

TYSA00	8000	system attribute type <base>
TYSACO	8001	color attribute
TYSABL	8002	blend attribute
TYSAHA	8003	hatch attribute
TYSADN	8004	density attribute
TYSAPL	8005	plines attribute
TYSAHU	8006	Bezier hull attribute
TYSARG	8013	regions attribute
TYSARF	8014	reflectivity attribute
TYSATR	8015	translucency attribute
TYSANM	8017	name
TYSABE	8018	V5 blend attribute
TYSAFG	8019	FG not found attribute
TYSADF	8020	deleted rubber faces
TYSAPH	8021	planar hatch attribute
TYSABN	8022	V9 blend attribute
TYSARD	8023	Region density
TYSAFD	8024	Face density
TYSAED	8025	Edge density
TYSAVD	8026	Vertex density
TYSARH	8027	Radial hatch attributes
TYSAUH	8028	Paramteric hatch attribute

User Attribute Type TYUA-- (9000+)

TYUA00	9000	user attribute type <base>

List Type TYLI-- (10000+)

TYLI00	10000	list type <base>
TYLIIN	10001	integer list
TYLIRL	10002	real list
TYLITG	10003	tag list

Feature Type TYFE-- (12000+)

TYFE00	12000	feature type <base>
TYFEFA	12001	face feature
TYFEED	12002	edge feature
TYFEVX	12003	vertex feature
TYFESU	12004	surface feature
TYFECU	12005	curve feature
TYFEPT	12006	point feature
TYFEMX	12007	mixed feature
TYFEIN	12008	instance feature
TYFERG	12009	region feature

Pseudo-type Owner TYOW-- (13000+)

TYOWNR	13000	pseudo-type owner <base>

Error Enquiry SLER-- (13100+)

SLER00	13100	error enquiry <base>
SLERRO	13101	routine
SLEREC	13102	error code
SLEREX	13103	explanation
SLERAR	13104	argument
SLERAI	13105	array index
SLERLE	13106	list entry
SLERTG	13107	tag

State Enquiry SLST-- (13200+)

SLST00	13200	state enquiry <base>
SLSTAR	13201	at rollmark
SLSTNF	13202	nsteps forward
SLSTNB	13203	nsteps back
SLSTVM	13204	virtual memory
SLSTFS	13205	free space
SLSTMT	13206	max tag

Local Operation Action SLLO-- (13300+)

SLLO00	13300	local op. action <base>
SLLOCP	13301	cap
SLLOGR	13302	grow

SLLOGP	13303	grow from parent
SLLORB	13304	leave rubber
SLLOGS	13305	grow or shrink
SLLOLI	13306	heal loops independently
SLLOLT	13307	heal loops together

Local Operation Return RTLO-- (13400+)

RTLO00	13400	local op. return <base>
RTLOOK	13401	body is ok
RTLONG	13402	body was negative
RTLOSX	13403	self-intersecting

File Enquiry SLFI-- (13500+)

SLFI00	13500	file enquiry <base>
SLFIVN	13501	modeler version

Control Point Type SLCP-- (13600+)

SLCP00	13600	control point type <base>
SLCPBS	13601	bspline
SLCPBZ	13602	bezier
SLCPSP	13603	spline

Parametric Basis SLBA-- (13650+)

SLBA00	13650	parametric basis <base>
SLBAHE	13651	hermite
SLBABZ	13652	bezier
SLBAPY	13653	polynomial
SLBATA	13654	taylor series

Simplification Level SLLE-- (13680+)

SLLE00	13680	simplification level <base>
SLLEGL	13681	global simplification
SLLELO	13682	local simplification

Pick Return SLPK-- (13690+)

SLPKOO	13690	pick return <base>
SLPKIR	13691	pick along infinite ray
SLPKSR	13692	pick along semi infinite ray

Parametric Prop PAPR-- (13700+)

PAPR00	13700	parametric prop <base>
PAPRPE	13701	periodic
PAPRNP	13702	non periodic
PAPRCS	13703	clamped start
PAPRCE	13704	clamped end
PAPRTL	13705	clamped top left twist vec
PAPRTR	13706	clamped top right twist vec
PAPRBL	13707	clamped bottom left twist
PAPRBR	13708	clamped bottom right twist
PAPRDS	13709	degenerate start
PAPRDE	13710	degenerate end
PAPRAM	13711	amalgamate knot vectors
PAPRKT	13712	knot vector supplied
PAPRNS	13713	natural start
PAPRNE	13714	natural end
PAPRUP	13715	u-parameter
PAPRVP	13716	v-parameter
PAPRPU	13717	periodic in u
PAPRPV	13718	periodic in v
PAPRIS	13719	insert null seg in start cu
PAPRIE	13720	insert null seg in end cu
PAPRCU	13721	force cubic lofting
PAPREX	13722	exact representation
PAPRNB	13723	natural bottom
PAPRNT	13724	natural top
PAPRNL	13725	natural left
PAPRNR	13726	natural right
PAPRCB	13727	clamped bottom
PAPRCT	13728	clamped top
PAPRCL	13729	clamped left
PAPRCR	13730	clamped right
PAPRKU	13731	u knot vector supplied
PAPRKV	13732	v knot vector supplied
PAPRIF	13733	infinite
PAPRXT	13734	extendable
PAPRNX	13735	not extendable
PAPRDP	13736	periodic, not cont. diff
PAPRCN	13737	continuous
PAPRDC	13738	discontinuous

PAPRLI	13739	linear
PAPRCI	13740	circular
PAPRDG	13741	degenerate
PAPRSD	13742	derv. start curve supplied
PAPRED	13743	derv. end curve supplied
PAPRSW	13744	degen. start curve supplied
PAPREW	13745	degen. end curve supplied
PAPRBC	13746	bounds coincident

Return State RTST-- (13800+)

RTST00	13800	return state <base>
RTSTNG	13801	body is inside out
RTSTCR	13802	data structure corrupt
RTSTMG	13803	missing geometry
RTSTSX	13804	self-intersecting topology
RTSTGX	13805	self-intersecting geometry
RTSTDG	13806	degenerate geometry
RTSTIN	13807	inconsistent geom & topol
RTSTIG	13808	invalid geometry
RTSTSZ	13809	size settings differ
RTSTBX	13810	size box violation
RTSTCF	13812	failure in checking attempt
RTSTWG	13813	withdrawn geometry types
RTSTMD	13814	consistency mending enacted
RTSTIO	13815	body was inside out
RTSTFF	13816	face-face inconsistency
RTSTOC	13817	open curve on ring edge
RTSTVC	13818	vertex not on curve
RTSTER	13819	edge reversed
RTSTSP	13820	SP-curves of edge not within edge tolerance
RTSTVT	13821	vertices touch
RTSTVS	13822	vertex not on surface
RTSTES	13823	edge not on surface
RTSTEO	13824	edges incorrectly ordered at vertex
RTSTMV	13825	missing vertex at surface singularity
RTSTLC	13826	loops inconsistent
RTSTGC	13827	geometry not G1-continuous
RTSTSH	13828	inconsistent shells
RTSTFC	13829	checker failure during face-face check
RTSTEF	13930	illegal edge-face intersection
RTSTEE	13831	illegal edge-edge intersection

RTSTFO	13832	faces out of order around edge
RTSTSG	13833	shell geometry and topology inconsistent
RTSTAC	13834	acorn shell clashes with another shell
RTSTRS	13835	regions of body are inconsistent
RTSTID	13836	invalid or duplicate identifiers
RTSTON	13837	open nominal geometry
RTSTVN	13838	vertex not on nomimal geometry
RTSTSN	13839	SP curves of edge not within tolerance of nominal geometry
RTSTRN	13840	nominal geometry in wrong direction for edge

Standard Rep Opt SROP-- (13900+)

SROP00	13900	standard rep opt <base>
SROPCU	13901	output cubics
SROPNR	13902	output non-rationals
SROPBS	13903	use a B-spline sf approx
SROPCT	13904	supply curve tolerance
SROPCN	13905	loops unconfined and not continuous
SROPCY	13906	loops confined and not continuous
SROPCC	13907	loops confined, continuous, closed
SROPCP	13908	loops confined, continuous, closed, exactly one periphery
SROPID	13909	include degeneracies in trim curves
SROPED	13910	exclude degeneracies from trim curves
SROPNG	13911	output associated geometry
SROPNT	13912	output associated topology
SROPNE	13913	don't extend surface to fit SP-curves

Masspr Option MAOP-- (14000+)

MAOP00	14000	masspr option <base>
MAOPNA	14001	no amount properties
MAOPAM	14002	amount and mass
MAOPCG	14003	center of gravity
MAOPIN	14004	moment of inertia
MAOPNP	14005	no periphery data
MAOPPE	14006	periphery required
MAOPNE	14007	no error estimates
MAOPEM	14008	modulus of errors given
MAOPEI	14009	error intervals given
MAOPCS	14010	treat entity members as complete solid

Output Format OUFO-- (14100+)

OUFO00	14100	output format <base>
OUFOPV	14101	position vector
OUFODR	14102	unit direction
OUFOAX	14103	axis: base + direction
OUFONP	14104	null position
OUFOCU	14120	curve pointer
OUFOSU	14121	surface pointer

Attribute_def Opt ATOP-- (14200+)

ATOP00	14200	attribute_def opt <base>
ATOPOW	14201	legal owner type codes
ATOPFL	14202	field types
ATOPCL	14203	class codes

Mending Option MDOP-- (14300+)

MDOP00	14300	mending option <base>
MDOPMD	14301	consistency mend
MDOPRB	14302	rubberize stranded topology
MDPONG	14304	negate inside-out bodies

Bulletin Board Event BBEV-- (14400+)

BBEV00	14400	bulletin board event <base>
BBEVCR	14401	create event
BBEVDE	14402	delete event
BBEVCH	14403	change event
BBEVSP	14404	split event
BBEVME	14405	merge event
BBEVTR	14406	transfer event
BBEVCO	14407	copy event
BBEVTF	14408	transform event
BBEVAC	14409	attribute change event

Bulletin Board Option BBOP-- (14500+)

BBOP00	14500	bulletin board option <base>
BBOPOF	14501	switch off
BBOPON	14502	switch on for tags
BBOPUF	14503	switch on for tags and user fields

Curve Intersect Clsf CICL-- (14610+)

CICL00	14610	curve intersect clsf <base>
CICLSI	14611	simple intersection
CICLTG	14612	tangency
CICLSC	14613	start of coincidence
CICLEC	14614	end of coincidence

Surface Intersect Clsf SICL-- (14650+)

SICL00	14650	surface intersect classification <base>
SICLSI	14651	simple intersection
SICLTG	14652	tangency
SICLBC	14653	boundary of region of coincidence

Closest Approach Opt CLOP-- (14700+)

CLOP00	14700	closest approach opt <base>
CLOPPT	14701	specify point close to soln
CLOPPR	14702	specify parameter estimates
CLOPUP	14703	specify upper distance bound
CLOPLW	14704	specify lower distance bound
CLOPTL	14705	specify distance tolerance
CLOPB1	14706	supply param box - 1st entity
CLOPB2	14707	supply param box - 2nd entity
CLOPP1	14708	supply param estimate - 1st entity
CLOPP2	14709	supply param estimate - 2nd entity
CLOPFA	14710	find all local minima

Curve Face Clsfction CFCL-- (14800+)

CFCL00	14800	curve face clsfction <base>
CFCLSI	14801	simple intersection
CFCLTG	14802	touch intersection
CFCLEF	14803	curve entering face
CFCLLF	14804	curve leaving face
CFCLEB	14805	curve entering boundary
CFCLLB	14806	curve leaving boundary
CFCLEI	14807	curve entering interior
CFCLLI	14808	curve leaving interior
CFCLTI	14809	tangent to inside of edge
CFCLTO	14810	tangent to outside of edge
CFCLUC	14811	unclassified

CFCLSC	14812	curve enters at start of coi
CFCLEC	14813	curve leaves at end of coi

Imprinting Opt IMOP-- (14900+)

IMOP00	14900	imprinting opt <base>
IMOPNT	14901	no imprinting on tool
IMOPOA	14902	imprint bounds of overlap
IMOPEF	14903	extend face list on target

Identify Region Opt IDOP-- (15000+)

IDOP00	15000	identify region opt <base>
IDOPUN	15001	simulated unite
IDOPIN	15002	simulated intersect
IDOPSU	15003	simulated subtract
IDOPFS	15004	selected facesets

CRTOBY Returns RTTO-- (15100+)

RTTO00	15100	CRTOBY returns <base>
RTTOOK	15101	input is OK
RTTOBB	15102	bad body id
RTTODE	15103	duplicate entry
RTTOUC	15104	undefined child
RTTODC	15105	duplicate child
RTTOWC	15106	wrong type of child
RTTOFC	15107	too few children
RTTOMC	15108	too many children
RTTOWP	15109	wrong type of parents
RTTOFP	15110	too few parents
RTTOMP	15111	too many parents
RTTODW	15112	disconnected wire
RTTOIL	15113	invalid loop
RTTOCS	15114	connected shells
RTTODS	15115	disjoint shell
RTTONM	15116	non-manifold vertex

Body Types BYTY-- (15200+)

BYTY00	15200	body types <base>
BYTYSO	15201	solid body
BYTYSH	15202	sheet body

BYTYWR	15203	wire body
BYTYMN	15204	minimum body
BYTYSS	15205	solid or sheet body

Discontinuities PADI-- (15300+)

PADI00	15300	discontinuities <base>
PADIG1	15301	G1 discontinuities

Closest Approach Return RTCL-- (15400+)

RTCL00	15400	closest approach return <base>
RTCLNO	15401	non-orthogonal to entity
RTCLPD	15402	positive distance from entity
RTCLND	15403	negative distance from entity
RTCLON	15404	on entity - zero distance
RTCLRS	15405	regional (non-unique) solution
RTCLLB	15406	distance less than lower bound
RTCLUB	15407	distance greater than upper bound

Mending Return RTMD (15500+)

RTMD00	15500	mending return <base>
RTMDMS	15501	mending successful
RTMDMF	15502	mend failed

Mending Return MDFA-- (15600+)

MDFA00	15600	mending return <base>
MDFAFE	15601	faulty edge
MDFANI	15602	non intersecting geometry
MDFAFM	15603	failure during mend
MDFARF	15604	rubber face
MDFACS	15605	coincident surfaces
MDFANS	15606	non intersecting surfaces

Knitting Return RTKN-- (15700+)

RTKN00	15700	knitting return <base>
RTKNOK	15701	knit successful
RTKNIN	15702	knit incomplete

CRBYGE Option CBOP-- (15800+)

CBOP00	15800	CRBYGE option <base>
CBOPUR	15801	U parameter range
CBOPVR	15802	V parameter range

BOPBYS Option BOOP-- (15900+)

BOOP00	15900	bopbys option <base>
BOOPIN	15901	intersect
BOOPSU	15902	subtract
BOOPUN	15903	unite
BOOPEF	15904	extend facelist
BOOPEC	15905	exclude boundary regions
BOOPIC	15906	include boundary regions
BOOPME	15907	merge
BOOPSX	15908	stop self intersection
BOOPTS	15909	trim with sheet
BOOCSH	15910	none of the instanced tools clash with each other
BOOINF	15911	none of the instanced tools clash with outer loop of targ
BOOCLP	15912	list of loops on target face that need to be tested
BOOPPS	15913	prune solid regions of the result
BOOPPV	15914	prune void regions of the result
BOOPPU	15915	punch sheet

Blending (v5) Option BLEC-- (16000+)

BLEC00	16000	blecre option <base>
BLECRI	16001	draw ribs
BLECDF	16002	draw/fix
BLECPR	16003	propagate
BLECTL	16004	set tolerance
BLECLI	16005	linear radius variation
BLECSM	16006	smooth
BLECCL	16007	cliff_edge
BLECCR	16008	circular cross-section
BLECCH	16009	linear cross-section
BLECSC	16010	same convexity cliff overflow
BLECEC	16011	end of edge cliff overflows
BLECNS	16012	no smooth overflows
BLECNC	16013	no cliff edge overflows
BLECNN	16014	no notch overflows

Blending (v5) Return BLCC-- (16050+)

BLCC00	16050	blechk option <base>
BLCCSN	16051	ends at singularity
BLCCOT	16052	unsupported old type
BLCCMX	16053	vertex too complex
BLCCRS	16054	adjoining face is rubber
BLCCRE	16055	truncating face is rubber
BLCCTV	16056	illegal two edge vertex
BLCCHM	16057	edge unsuitable for chamfer
BLCCXT	16058	require extension of B-surf
BLCCIR	16059	inconsistent ranges
BLCCIT	16060	inconsistent types
BLCCAB	16061	adjoining edge not blended
BLCCOL	16062	completely overlaps edge loop
BLCCOB	16063	overlapping blends
BLCCOU	16064	overlaps unblended edge
BLCCUN	16065	unspecified numerical problem
BLCCUE	16066	unspecified problem at end
BLCCRL	16067	range too large
BLCCOE	16068	illegal overlap at end
BLCCIE	16069	illegal end boundary
BLCCIX	16070	cannot intersect chamfers
BLCCEX	16071	end overlaps unblended edge
BLCCOI	16072	illegal blend on another edge
BLCCTN	16073	on tangent edge
BLCCIP	16074	inconsistent cliffedge parameters

Intersection Param INOP-- (16100+)

INOP00	16100	insusu option <base>
INOPBX	16101	Box of intersection supplied
INOPPF	16102	Parameter box of intersection supplied for surface/face 1
INOPPS	16103	Box of intersection supplied for surface/face 2
INOPSI	16104	Return all intersections through given point
INOPBP	16105	Return single intersection between given 2 points

Surface Rev Option CROP-- (16200+)

CROP00	16200	crrvsu option <base>
CROPPR	16201	supply parameter range
CROPSI	16202	Simplify surface

Trimmed Surface Option TSOP-- (16300+)

TSOP00	16300	trimmed surface check option <base>
TSOPWR	16301	check for wire topology
TSOPSX	16302	check for self-intersections
TSOPLC	16303	check loops for consistency

Sheet Body State RTTS-- (16400+)

RTTS00	16400	trimmed surface state code <base>
RTTSOK	16401	all checks successful
RTTSFR	16402	redundant face with respect to tolerances
RTTSCI	16403	loops of curves inconsistent directions
RTTSSX	16404	edges intersect at position other than vertex
RTTSLI	16405	invalid loop combination for surface type
RTTSEO	16406	edges incorrectly ordered at vertex

Connected Entities Type IDTY-- (16500+)

IDTY00	16500	common connection type <base>
IDTYCS	16501	common curves of surfaces
IDTYSC	16502	common surfaces of curve
IDTYEF	16503	common edges of faces

Checking Option CHOP-- (16600+)

CHOP00	16600	chcken option <base>
CHOPCR	16601	check for corrupt datastructure
CHOPIG	16602	check for invalid or self-intersecting geometry
CHOPED	16603	check for inconsistencies in edges
CHOPFA	16604	check for inconsistencies in faces
CHOPSX	16605	check for self-intersecting faces
CHOPLC	16606	check for loop consistency of faces
CHOPBX	16607	check for size-box violations
CHOPFF	16608	check for face-face inconsistencies
CHOPSH	16609	check for inside-out or inconsistent shells
CHOPPV	16610	force self-intersection tests on Pre-V5 b-geometry
CHOPNO	16611	no options, force all appropriate checks to the geometry

Trim Sheet Option SLTR-- (16700+)

SLTR00	16700	trimsh option <base>
SLTRKE	16701	select regions to remain on the sheet

SLTRRE	16702	select regions to be deleted from the sheet
SLTRTL	16703	enable closing of loop gaps to optional tolerance

SHAREN Option SHOP-- (16800+)

SHOP00	16800	sharen option <base>
SHOPIC	16801	only process local intersection curve relationships

Local Checking Level LOCH-- (16900+)

LOCH00	16900	local checking level <base>
LOCHNC	16901	no local checking
LOCHFA	16902	local face checking only
LOCHFC	16903	full local checking including face-face checking

Offset Operations Return RTOF-- (17000+)

RTOF00	17000	offset operations return <base>
RTOFOK	17001	body is OK
RTOFSO	17002	surface failed to offset
RTOFVM	17003	vertex was not modified
RTOFEM	17004	edge was not modified
RTOFTL	17005	supplied edge tolerance too large
RTOFVT	17009	vertices of edge touched
RTOFNG	17013	offset body was negative
RTOFFA	17014	face checking failed
RTOFSX	17015	self-intersecting, face-face checking failed
RTOFED	17016	edge degenerates
RTOFSS	17017	failed to find the side surface
RTOFSC	17018	failed to find the side curve

ENPIFA Option PFOP-- (17100+)

PFOP00	17100	enpifa option <base>
PFOPLO	17101	only consider specified loops

RETLEN Status RTTL-- (17200+)

RTTL00	17200	retlen status <base>
RTTLOK	17201	replace tolerance successful
RTTLNT	17202	tolerance not altered at near tangency
RTTLMG	17203	could not replace tol due to missing geometry
RTTLRF	17204	re-computation of edge geometry failed

TRSHCU Option TRSH-- (17300+)

TRSH00	17300	trshcu option <base>
TRSHPD	17301	project curves in given direction
TRSHTR	17302	trim as well as imprint
TRSHLC	17303	keep region to left of first curve
TRSHRC	17304	keep region to right of first curve
TRSHIL	17305	keep region inside closed loop
TRSHOL	17306	keep region outside closed loop

BLEFXF Option FXFT-- (17400+)

FXFT00	17400	blefxf option <base>
FXFTNT	17401	do not trim blend
FXFTTB	17402	trim blend to walls
FXFTTW	17403	trim blend and walls
FXFTAT	17404	trim blend and walls and attach blend
FXFTCB	17405	constant radius rolling ball blend
FXFTVB	17406	variable radius rolling ball blend
FXFTHL	17407	blend constrained by tangent hold lines
FXFTCE	17408	blend constrained by cliffedges
FXFTHP	17409	help point provided for blend
FXFTLP	17410	limit point provided for blend
FXFTPR	17411	blend may propagate outside walls
FXFTMS	17412	create multiple blends if possible
FXFTTL	17413	tolerance associated with blend
FXFTRC	17414	get rho values from law curve
FXFTEO	17415	allow notching
FXFTCL	17416	blend constrained by conic hold lines
FXFTSO	17417	create solid body if possible
FXFTCC	17418	blend curvature continuous at hold lines
FXFTDB	17419	disc blend
FXFTST	17420	short trim blend to walls
FXFTLT	17421	long trim blend to walls

BLEFXF Error FXFE-- (17450+)

FXFE00	17450	blefxf error <base>
FXFEOK	17451	face face blend succeeded
FXFEST	17452	failed to attach blend
FXFEER	17453	failed to create blend
FXFEID	17454	insufficient blend data

FXFEXD	17455	inconsistent blend data
FXFEIF	17456	invalid side wall
FXFEIR	17457	invalid blend radius definition
FXFEIH	17458	invalid tangent hold line data
FXFEIC	17459	invalid cliffedge data
FXFEFC	17460	face too tightly curved
FXFERS	17461	blend radius is too small
FXFERL	17462	blend radius is too large
FXFELN	17463	left wall normal is wrong
FXFERN	17464	right wall normal is wrong
FXFEBN	17465	both wall normals are wrong
FXFESC	17466	blend sheets intersect
FXFEWC	17467	walls clash
FXFEGX	17468	blend face has self-intersecting geometry
FXFEFF	17469	blend has resulted in face-face inconsistency
FXFERV	17470	invalid rho values in law curve
FXFEAR	17471	ranges too asymmetric for geometry
FXFECL	17472	invalid conic hold line data
FXFEIS	17473	invalid spine data

Kernel Interface Error Codes B

B.1 Introduction

This appendix lists all the error codes used by the kernel interface, in both numeric and alphabetic order.

B.2 KI error codes in numeric order

KI_no_errors	0	Operation was successful
KI_bad_angle	1	angle out of range
KI_buffer_overflow	2	array too small to hold results
KI_radii_both_0	3	both radii are zero
KI_cone_too_sharp	4	cone cannot be distinguished from a cylinder
KI_has_no_name	7	entity has no name
KI_has_no_owner	8	entity has no owner so cannot be named
KI_wrong_entity	10	entity is not of type expected
KI_bad_name	11	name is invalid
KI_bad_type_combn	12	invalid combination of types
KI_not_unique	13	not unique
KI_distance_lt_0	14	distance is negative
KI_distance_le_0	15	distance is zero or negative
KI_radius_le_0	16	radius is zero or negative
KI_radius_lt_0	18	radius is negative
KI_not_found	19	specified entity not found
KI_not_connected	20	no such connection
KI_not_a_tag	22	tag is invalid or dead
KI_null_axis	25	axis vector has zero length
KI_cant_open_jrnl	27	journal file could not be opened
KI_has_parent	28	entity already has a parent
KI_bad_index	29	invalid index
KI_bad_type	30	invalid type
KI_null_direction	31	direction vector has zero length
KI_rot_angle_eq_0	32	zero angle rotation asked for
KI_lt_3_sides	33	number of sides is less than 3
KI_is_attached	34	geometric entity is attached to topology
KI_dont_intersect	35	no intersection

KI_majaxi_not_perpn	36	major axis and axis not perpendicular
KI_wrong_transf	37	transformation is unsuitable for operation
KI_bad_selection_code	38	invalid selection code
KI_bad_value	39	value is not as expected
KI_sc_factor_le_0	40	scale must be greater than zero
KI_su_are_coincident	41	coincident surfaces
KI_bulletinb_is_off	42	bulletin board is not active
KI_none_mergeable	48	no mergeable entities
KI_cant_do_tweak	50	tweak cannot be performed
KI_inconsistent_geom	51	geometry is inconsistent with neighbors
KI_not_on_face	54	not within resolution distance of face
KI_impossible_swing	55	cannot determine swung geometry
KI_impossible_sweep	57	cannot determine swept geometry
KI_key_not_found	58	key not found
KI_not_in_same_part	59	entities do not all belong to the same part
KI_no_geometry	61	no geometry attached
KI_geom_topol_mismatch	62	geometry/topology mismatch
KI_receive_failed	63	receive failed
KI_geom_not_needed	64	topological entity already owns geometry
KI_not_on_curve	67	not within resolution distance of curve
KI_still_referenced	68	argument is still referenced
KI_fragment	73	operation would cause sheet to break apart
KI_cant_find_su	77	unable to find a surface
KI_empty_list	79	empty list not valid for this operation
KI_not_a_list	80	a list was not provided when one was required
KI_mass_eq_0	82	bodies have zero total mass
KI_density_le_0	85	a body has zero or negative density
KI_dont_make_solid	87	unable to make solid from faces
KI_missing_geom	96	a topological entity lacks geometry
KI_attr_not_found	99	attribute of required type not found
KI_not_solid	101	a body is not solid
KI_corrupt_body	103	model data structure is corrupt
KI_bad_geom_topol	105	geometry inconsistent with topology
KI_negative_body	106	body is inside out
KI_bad_char_string	109	invalid character string
KI_bad_spec_code	110	specification code is out of range
KI_weight_le_0	111	weights are not all strictly positive
KI_illegal_degeneracy	116	illegal degeneracy
KI_bad_parameter	120	parameter out of range
KI_discontinuous_surface	129	adjacent patches do not meet
KI_discontinuous_curve	131	adjacent curve segments do not meet

KI_order_lt_2	132	order is less than 2
KI_bad_dimension	135	invalid dimension
KI_su_self_intersect	141	would produce a self_intersecting surface
KI_cant_do_intersect	157	unable to perform requested intersection
KI_cant_fix_blends	330	could not fix blends in body
KI_bad_blend_bound	334	illegal blend boundary
KI_not_blended	335	no blend on edge
KI_blend_didnt_check	336	blend fixed but failed checker
KI_bad_request_code	350	invalid request code
KI_wrong_entity_in_list	357	entity of wrong type in list
KI_not_same_length	359	lists should be the same length but are not
KI_bad_view_mx	360	invalid view matrix
KI_bad_pixel_map	361	mapping from model to pixel-space is invalid
KI_bad_light_source	364	invalid light-source definition
KI_eye_in_box	367	eye-point may not be inside box of entity
KI_cyclic_assy	503	operation would cause cyclic reference
KI_anon_sub_part	504	instance of anonymous sub-part of stored part
KI_different_types	505	entities in list are not all of same type
KI_existing_attr_type	506	attribute type already defined
KI_majrad_minrad_mismatch	507	incompatible values for major and minor radii
KI_radius_sum_le_0	508	majrad < zero and majrad + minrad <= zero
KI_wrong_list_type	509	list is not of the expected type
KI_bad_tag_in_list	510	list contains dead or invalid tag
KI_duplicate_list_item	511	list contains duplicate item
KI_not_in_feat	512	entity not found in feature
KI_wrong_type_for_feat	513	entity type is inconsistent with feature type
KI_list_too_short	519	not enough items in list
KI_already_in_feat	520	entity is already in feature
KI_attr_mismatch	522	attribute properties inconsistent with defn
KI_list_wrong_length	523	list not of expected length or is too long
KI_part_not_keyed	524	part is not keyed and cannot be unloaded
KI_cant_heal_wound	525	cannot heal wound - impossible geometry
KI_already_loaded	528	part with key already loaded
KI_already_saved	529	part is already stored in archive
KI_key_in_use	530	key already used in archive
KI_closed_faces	531	cannot make solid from closed set of faces
KI_at_singularity	532	coords at singularity of surface
KI_size_mismatch	533	archived part has different size settings
KI_duplicate_tools	540	duplication in list of tool bodies
KI_instanced_tools	541	instanced tool bodies
KI_mixed_sheets_solids	542	mixture of sheet and solid tool bodies

KI_cant_unite_solid_sheet	543	cannot unite solid with sheet
KI_same_tool_and_target	545	target body cannot also be a tool body
KI_invalid_bodies	546	boolean failure or invalid bodies
KI_non_manifold	547	non-manifold body or boundary
KI_t_sheet	549	T-sheet
KI_wrong_sub_type	553	sub-type of entity is unsuitable
KI_attr_defn_mismatch	555	archived attribute defns don't match current
KI_cant_find_file	557	cannot find file
KI_get_snapshot_failed	558	failed to restore snapshot
KI_transmit_failed	560	transmit failed
KI_bad_filename	561	character string is not a valid file name
KI_save_snapshot_failed	562	failed to save snapshot
KI_bad_key	565	key has invalid syntax
KI_journal_not_open	566	journal file not open
KI_bad_state_combn	570	new state is incompatible with original state
KI_rollmark_failed	850	failed to set rollmark
KI_no_rollmark	854	no previous roll-mark exists
KI_roll_is_off	855	logging for roll-back is disabled
KI_roll_forward_fail	856	roll forward not possible
KI_impossible_taper	860	taper cannot be performed
KI_system_error	900	modeler error: please report fault
KI_memory_full	901	modeler has run out of virtual memory
KI_nitems_lt_0	902	number of items is negative
KI_nitems_le_0	903	number of items is zero or negative
KI_modified_sub_part	904	part has modified sub-part
KI_part_not_isolated	905	part not isolated
KI_null_arg_address	906	argument address given as zero
KI_bad_option_data	907	option data not valid for relevant option
KI_not_a_logical	908	value is not KI_true (1) or KI_false (0)
KI_bad_box	909	box is too small or too large
KI_bad_position	911	position lies outside modeler resolution
KI_empty_assy	912	assembly instances no bodies
KI_keyed_part_mismatch	913	archived part not same type as part in memory
KI_unsuitable_entity	914	entity unsuitable for requested operation
KI_not_on_surface	915	not within resolution distance of surface
KI_bad_shared_entity	916	entity would become illegally shared
KI_bad_shared_dep	917	dependent of entity would be illegally shared
KI_attr_type_not_defined	919	attribute type not defined
KI_bad_blend_param	920	blend parameter out of range
KI_bad_sharing	921	sharing prohibits operation
KI_corrupt_file	922	invalid file contents

KI_wrong_version	923	file is incompatible with this version
KI_not_at_rollmark	924	state of model changed since last roll-mark
KI_radius_eq_0	925	radius is zero or very near to zero
KI_radius_too_large	926	radius is too large for modeler resolution
KI_distance_too_large	927	distance too large for modeler resolution
KI_cant_open_file	928	failed to open file
KI_at_terminator	929	coords at terminator of curve
KI_bad_precision	930	precision does not lie within allowed range
KI_modeller_not_started	931	STAMOD must be first call to KI
KI_modeller_not_stopped	932	calling STAMOD twice without calling STOMOD
KI_bad_user_field_size	933	specified user-field size not in range 0 - 16
KI_recursive_call	934	calling KI before the previous call completes
KI_bad_hull	935	Bezier hull is too complicated
KI_usfd_mismatch	936	file has wrong user-field size
KI_wrong_format	937	trying to open file in wrong format
KI_wire_body	938	wire or acorn body is unsuitable for this operation
KI_not_sheet	939	body is not a sheet
KI_bad_wire	940	invalid wire would result
KI_bad_end_points	941	curve not defined between end points
KI_crossing_edge	942	curve crosses edge
KI_crossing_vertex	943	curve crosses vertex
KI_bad_vertex	944	too many edges at vertex
KI_aborted	945	operation aborted following an interrupt
KI_not_interrupted	946	kernel has not been interrupted
KI_run_time_error	947	non fatal run-time error
KI_fatal_error	948	irrecoverable run-time error
KI_no_user_fields	949	user fields are not in use
KI_wrong_surface	950	wrong type of surface for operation
KI_opposed_sheets	951	attempt to unite opposed sheets
KI_coplanar	952	directions are coplanar
KI_bad_accuracy	956	accuracy out of range
KI_coincident	957	start and end coincide but curve not closed
KI_atol_too_small	958	invalid angular control set
KI_ctol_too_small	959	invalid chordal control set
KI_stol_too_small	960	invalid step length control set
KI_wrong_direction	961	start and end in wrong order
KI_non_orth_matrix	962	non-orthogonal matrix
KI_bad_component	963	array positions 12, 13, 14 must be zero
KI_bad_rollfile_size	964	size for rollback file too large or too small

KI_cant_be_aborted	965	roll-operations cannot be aborted
KI_hulls_intersect	966	invalid intersection between convex hulls
KI_abort_from_go	967	a GO routine returned status-code <abort>
KI_all_faces_in_body	969	not valid for all faces in the body
KI_schema_access_error	970	open/close/read/write error for schema file
KI_schema_corrupt	971	contents of schema file not as expected
KI_cant_intsc_solid_sheet	972	cannot intersect solid target with sheet tool
KI_file_access_error	973	unexpected file-access error from Frustrum
KI_bad_file_format	974	invalid value given for file format
KI_bad_file_guise	975	invalid value given for file guise
KI_bad_rolling_ball	976	rolling ball blend is not allowed on edge
KI_coincident_points	977	Coincident control or spline points
KI_bad_knots	978	Invalid knot-vector
KI_bad_derivative	979	derivative vector too big
KI_wrong_number_knots	980	Knot vector is too long or too short
KI_wrong_number_derivs	981	Too many or too few derivatives supplied
KI_incompatible_props	982	Combination of properties is impossible or ambiguous
KI_repeated_knots	983	Repeated knots in knot vector invalid in this context
KI_curves_dont_meet	984	Curves are not sequent
KI_insufficient_curves	985	Too many or too few curves
KI_bad_curves	986	Curves invalid for this operation
KI_bad_order	987	Invalid order for operation
KI_insufficient_points	988	Insufficient control or spline points
KI_bad_parametric_prop	989	Bad parametric property code
KI_illegal_owner	990	Specified owner is inappropriate or invalid
KI_unchecked_entity	991	Entity unchecked and may be invalid for this operation
KI_incompatible_curves	992	Curves not compatible for this operation
KI_cant_make_bspline	993	Failure to represent entity in B-spline form
KI_cu_are_coincident	994	Coincident curves
KI_withdrawn_surface	995	Part contains a withdrawn blend surface
KI_face_not_planar	996	Can only find cofg/inertia for planar faces
KI_request_not_supported	997	Requested mass properties not implemented
KI_contradictory_request	998	Contradictory or unknown requests for mass properties
KI_invalid_geometry	999	Geometry fails to pass checks
KI_file_already_exists	1000	Can't create file of same name as existing one
KI_too_many_control_pts	1001	parametric entity has too many control points
KI_bad_string	1002	string is invalid
KI_mend_attempt_failure	1003	mending attempt has failed

KI_bad_tag_in_list_tree	1004	list tree contains dead or invalid tag
KI_bad_list_tree	1005	lists do not form a valid tree
KI_cyclic_list_reference	1006	lists refer to each other cyclically
KI_empty_list_in_tree	1007	empty list found in list tree
KI_cant_make_trimmed_sf	1008	failed to make trimmed surface
KI_bad_entity_event_comb	1009	bad combination of entity/event
KI_too_many_derivatives	1010	too many derivatives requested
KI_bad_deriv_vertices	1011	bad vertex list for derivative curve
KI_bad_degen_vertices	1012	bad vertex list for degenerate curve
KI_not_on_edge	1013	not within resolution distance of edge
KI_closest_approach_failed	1014	failed to find closest approach
KI_cant_do_clash	1015	clash failure
KI_targ_faces_many_bodies	1016	target faces in list are from more than one body
KI_tool_faces_many_bodies	1017	tool faces in list are from more than one body
KI_cant_do_imprint	1018	imprint failure
KI_topol_not_from_body	1019	topology is not from expected body
KI_inconsistent_facesets	1020	failure to identify facesets
KI_FG_evaluator_not_found	1021	foreign geometry evaluator does not exist
KI_FG_data_alloc_error	1022	foreign geometry evaluator space allocation fault
KI_FG_data_not_found	1023	could not access foreign geometry data
KI_FG_evaluator_error	1024	foreign geometry evaluator failure
KI_FG_modelling_error	1025	cannot model with instance of foreign geometry
KI_solid_body	1026	solid body is unsuitable for this operation
KI_different_bodies	1027	vertices on different bodies
KI_wrong_number_edges	1028	only 2 edges at vertex permissible
KI_cant_blend_vertex	1029	could not blend vertices as requested
KI_blends_overlap	1030	blends would overlap with each other
KI_edges_intersect	1031	edges would intersect
KI_not_in_same_body	1032	entities do not all belong to same the body
KI_unsuitable_topology	1033	topology is unsuitable
KI_cu_self_intersect	1034	curve self-intersects
KI_linear_multi_seg	1035	linear B-spline with >1 seg not allowed
KI_no_eds_from_target	1036	no knitting pattern edges form target
KI_cant_offset	1037	underlying surface cannot be offset
KI_FG_real_data_error	1038	foreign geometry real data error
KI_FG_integer_data_error	1039	evaluator integer data error
KI_partial_coi_found	1040	failure due to detection of a partial coincidence
KI_bodies_dont_knit	1041	bodies have no coincident edges
KI_pattern_invalid	1042	pattern would produce invalid body
KI_bad_tolerance	1043	tolerance is less than Parasolid tolerance

KI_cant_extract_geom	1044	failure to extract necessary geometry to make body
KI_bad_basis_surf	1045	SP-curve not defined on supplied basis surface
KI_FG_receive_failure	1046	Archived part contains foreign geometry which fails to receive
KI_FG_snapshot_failure	1047	Snapshot contains foreign geometry which fails re-initialization
KI_cant_create_pattern	1048	Failed to create knitting pattern
KI_tag_limit_exceeded	1049	Tag limit would be exceeded
KI_tag_limit_out_of_range	1050	Invalid tag limit
KI_cant_find_extreme	1051	Failed to find extreme point
KI_disc_full	1052	Disc is full
KI_cant_find_derivs	1053	failed to find derivatives
KI_too_many_targets	1054	too many target bodies
KI_duplicate_targets	1055	duplicates in list of targets
KI_curve_already_trimmed	1056	attempting to trim a trimmed curve
KI_curve_too_short	1057	trimmed curve is shorter than linear resolution
KI_boolean_failure	1058	inconsistent arguments, or internal error
KI_duplicate_item	1059	duplicate item in option data
KI_failed_to_trim	1060	failed to trim
KI_unsuitable_loop	1061	loop is of wrong type
KI_failed_to_replace	1062	unable to replace surface of sheet
KI_failed_to_create_sp	1063	failed to create SP-curve
KI_tolerance_too_tight	1064	failed to meet tolerances
KI_fru_error	1065	Frustrum error
KI_incorrect_mc_conf	1066	machine configuration not authorized for Parasolid
KI_partial_no_intersect	1067	no imprinting in local boolean
KI_none_shared	1068	no shared geometry
KI_cant_hollow	1069	failed to hollow body
KI_not_in_same_shell	1070	entities not all from the same body
KI_general_body	1071	function not supported for general bodies
KI_bad_thickness	1072	thickness is zero
KI_non_smooth_edge	1073	normals discontinuous across edge
KI_degenerate_vertex	1074	degenerate vertex not allowed
KI_cant_thicken	1075	failed to thicken sheet
KI_crossing_face	1076	curve crosses face
KI_not_in_region	1077	not inside region
KI_empty_body	1078	empty general body
KI_sheet_untrimmed	1079	trim curves didn't remove any part of sheet
KI_fxf_blend_failed	1080	failed to create face-face blend
KI_fxf_blend_bad_token	1081	illegal face-face blend token

KI_file_read_corruption	1082	corrupt data read, perhaps an NFS problem
KI_trim_loop_degenerate	1083	Trimming loop degenerates at given tolerance
KI_solid_has_void	1084	Solid body contains void
KI_not_in_same_partition	1086	Entities are not all in the same partition
KI_instanced_body	1087	Body is instanced
KI_entity_not_new	1088	An entity was not created since the last roll operation
KI_applio_not_registered	1089	The transmit/receive i/o functions have not been registered
KI_more_than_one_part	1090	More than one part in transmit file
KI_bad_field_conversion	1091	Field size incompatible between versions

B.3 KI error codes in alphabetic order

KI_FG_data_alloc_error	1022	foreign geometry evaluator space allocation fault
KI_FG_data_not_found	1023	could not access foreign geometry data
KI_FG_evaluator_error	1024	foreign geometry evaluator failure
KI_FG_evaluator_not_found	1021	foreign geometry evaluator does not exist
KI_FG_integer_data_error	1039	foreign geometry integer data error
KI_FG_modelling_error	1025	cannot model with instance of foreign geometry
KI_FG_real_data_error	1038	foreign geometry real data error
KI_FG_receive_failure	1046	Archived part contains foreign geometry which fails to receive
KI_FG_snapshot_failure	1047	Snapshot contains foreign geometry which fails re-initialization
KI_abort_from_go	967	a GO routine returned status-code <abort>
KI_aborted	945	operation aborted following an interrupt
KI_all_faces_in_body	969	not valid for all faces in the body
KI_already_in_feat	520	entity is already in feature
KI_already_loaded	528	part with key already loaded
KI_already_saved	529	part is already stored in archive
KI_anon_sub_part	504	instance of anonymous sub-part of stored part
KI_applio_not_registered	1089	The transmit/receive i/o functions have not been registered
KI_at_singularity	532	coords at singularity of surface
KI_at_terminator	929	coords at terminator of curve
KI_atol_too_small	958	invalid angular control set
KI_attr_defn_mismatch	555	archived attribute defns don't match current
KI_attr_mismatch	522	attribute properties inconsistent with defn

KI_attr_not_found	99	attribute of required type not found
KI_attr_type_not_defined	919	attribute type not defined
KI_bad_accuracy	956	accuracy out of range
KI_bad_angle	1	angle out of range
KI_bad_basis_surf	1045	SP-curve not defined on supplied basis surface
KI_bad_blend_bound	334	illegal blend boundary
KI_bad_blend_param	920	blend parameter out of range
KI_bad_box	909	box is too small or too large
KI_bad_char_string	109	invalid character string
KI_bad_component	963	array positions 12, 13, 14 must be zero
KI_bad_curves	986	Curves invalid for this operation
KI_bad_degen_vertices	1012	bad vertex list for degenerate curve
KI_bad_derivative	979	derivative vector too big
KI_bad_deriv_vertices	1011	bad vertex list for derivative curve
KI_bad_dimension	135	invalid dimension
KI_bad_end_points	941	curve not defined between end points
KI_bad_entity_event_comb	1009	bad combination of entity/event
KI_bad_field_conversion	1091	Field size incompatible between versions
KI_bad_file_format	974	invalid value given for file format
KI_bad_file_guise	975	invalid value given for file guise
KI_bad_filename	561	character string is not a valid file name
KI_bad_geom_topol	105	geometry inconsistent with topology
KI_bad_hull	935	Bezier hull is too complicated
KI_bad_index	29	invalid index
KI_bad_key	565	key has invalid syntax
KI_bad_knots	978	Invalid knot-vector
KI_bad_light_source	364	invalid light-source definition
KI_bad_list_tree	1005	lists do not form a valid tree
KI_bad_name	11	name is invalid
KI_bad_option_data	907	option data not valid for relevant option
KI_bad_order	987	Invalid order for operation
KI_bad_parameter	120	parameter out of range
KI_bad_parametric_prop	989	Bad parametric property code
KI_bad_pixel_map	361	mapping from model to pixel-space is invalid
KI_bad_position	911	position lies outside modeler resolution
KI_bad_precision	930	precision does not lie within allowed range
KI_bad_request_code	350	invalid request code
KI_bad_rollfile_size	964	size for rollback file too large or too small
KI_bad_rolling_ball	976	rolling ball blend is not allowed on edge
KI_bad_selection_code	38	invalid selection code

KI_bad_shared_dep	917	dependent of entity would be illegally shared
KI_bad_shared_entity	916	entity would become illegally shared
KI_bad_sharing	921	sharing prohibits operation
KI_bad_spec_code	110	specification code is out of range
KI_bad_state_combn	570	new state is incompatible with original state
KI_bad_string	1002	string is invalid
KI_bad_tag_in_list	510	list contains dead or invalid tag
KI_bad_tag_in_list_tree	1004	list tree contains dead or invalid tag
KI_bad_thickness	1072	thickness is zero
KI_bad_tolerance	1043	tolerance is less than Parasolid tolerance
KI_bad_type	30	invalid type
KI_bad_type_combn	12	invalid combination of types
KI_bad_user_field_size	933	specified user-field size not in range 0 - 16
KI_bad_value	39	value is not as expected
KI_bad_vertex	944	too many edges at vertex
KI_bad_view_mx	360	invalid view matrix
KI_bad_wire	940	invalid wire would result
KI_blend_didnt_check	336	blend fixed but failed checker
KI_blends_overlap	1030	blends would overlap with each other
KI_bodies_dont_knit	1041	bodies have no coincident edges
KI_boolean_failure	1058	inconsistent arguments, or internal error
KI_buffer_overflow	2	array too small to hold results
KI_bulletinb_is_off	42	bulletin board is not active
KI_cant_be_aborted	965	roll-operations cannot be aborted
KI_cant_blend_vertex	1029	could not blend vertices as requested
KI_cant_create_pattern	1048	failed to create knitting pattern
KI_cant_do_clash	1015	clash failure
KI_cant_do_imprint	1018	imprint failure
KI_cant_do_intersect	157	unable to perform requested intersection
KI_cant_do_tweak	50	tweak cannot be performed
KI_cant_extract_geom	1044	failure to extract necessary geometry to make body
KI_cant_find_derivs	1053	failed to find derivatives
KI_cant_find_extreme	1051	failed to find extreme point
KI_cant_find_file	557	cannot find file
KI_cant_find_su	77	unable to find a surface
KI_cant_fix_blends	330	could not fix blends in body
KI_cant_heal_wound	525	cannot heal wound - impossible geometry
KI_cant_hollow	1069	failed to hollow body
KI_cant_intsc_solid_sheet	972	cannot intersect solid target with sheet tool
KI_cant_make_bspline	993	Failure to represent entity in B-spline form

KI_cant_make_trimmed_sf	1008	failed to make trimmed surface
KI_cant_offset	1037	underlying surface cannot be offset
KI_cant_open_file	928	failed to open file
KI_cant_open_jrnl	27	journal file could not be opened
KI_cant_thicken	1075	failed to thicken sheet
KI_cant_unite_solid_sheet	543	cannot unite solid with sheet
KI_closed_faces	531	cannot make solid from closed set of faces
KI_closest_approach_failed	1014	failed to find closest approach
KI_coincident	957	start and end coincide but curve not closed
KI_coincident_points	977	Coincident control or spline points
KI_cone_too_sharp	4	cone cannot be distinguished from a cylinder
KI_contradictory_request	998	Contradictory or unknown requests for mass properties
KI_coplanar	952	directions are coplanar
KI_corrupt_body	103	model data structure is corrupt
KI_corrupt_file	922	invalid file contents
KI_crossing_edge	942	curve crosses edge
KI_crossing_face	1076	curve crosses face
KI_crossing_vertex	943	curve crosses vertex
KI_ctol_too_small	959	invalid chordal control set
KI_cu_are_coincident	994	Coincident curves
KI_cu_self_intersect	1034	curve self-intersects
KI_curve_already_trimmed	1056	attempting to trim a trimmed curve
KI_curve_too_short	1057	curve is shorter than linear resolution
KI_curves_dont_meet	984	Curves are not sequent
KI_cyclic_assy	503	operation would cause cyclic reference
KI_cyclic_list_reference	1006	lists refer to each other cyclically
KI_degenerate_vertex	1074	degenerate vertex not allowed
KI_density_le_0	85	a body has zero or negative density
KI_different_bodies	1027	vertices on different bodies
KI_different_types	505	entities in list are not all of same type
KI_disc_full	1052	disc is full
KI_discontinuous_curve	131	adjacent curve segments do not meet
KI_discontinuous_surface	129	adjacent patches do not meet
KI_distance_le_0	15	distance is zero or negative
KI_distance_lt_0	14	distance is negative
KI_distance_too_large	927	distance too large for modeler resolution
KI_dont_intersect	35	no intersection
KI_dont_make_solid	87	unable to make solid from faces
KI_duplicate_item	1059	duplicate item in option data
KI_duplicate_list_item	511	list contains duplicate item

KI_duplicate_targets	1055	duplicate in list of targets
KI_duplicate_tools	540	duplication in list of tool bodies
KI_edges_intersect	1031	edges would intersect
KI_empty_assy	912	assembly instances no bodies
KI_empty_body	1078	empty general body
KI_empty_list	79	empty list not valid for this operation
KI_empty_list_in_tree	1007	empty list found in list tree
KI_entity_not_new	1088	An entity was not created since the last roll operation
KI_existing_attr_type	506	attribute type already defined
KI_eye_in_box	367	eye-point may not be inside box of entity
KI_face_not_planar	996	Can only find cofg/inertia for planar faces
KI_failed_to_create_sp	1063	failed to create SP-curve
KI_failed_to_replace	1062	unable to replace surface of sheet
KI_failed_to_trim	1060	failed to trim
KI_fatal_error	948	irrecoverable run-time error
KI_file_access_error	973	unexpected file-access error from Frustrum
KI_file_already_exists	1000	Can't create file of same name as existing one
KI_file_read_corruption	1082	corrupt data read, perhaps an NFS problem
KI_fragment	73	operation would cause sheet to break apart
KI_fru_error	1065	Frustrum error
KI_fxf_blend_failed	1080	failed to create face-face blend
KI_fxf_blend_bad_token	1081	illegal face-face blend token
KI_general_body	1071	function not supported for general bodies
KI_geom_not_needed	64	topological entity already owns geometry
KI_geom_topol_mismatch	62	geometry/topology mismatch
KI_get_snapshot_failed	558	failed to restore snapshot
KI_has_no_name	7	entity has no name
KI_has_no_owner	8	entity has no owner so cannot be named
KI_has_parent	28	entity already has a parent
KI_hulls_intersect	966	invalid intersection between convex hulls
KI_illegal_degeneracy	116	illegal degeneracy
KI_illegal_owner	990	specified owner is inappropriate or invalid
KI_impossible_sweep	57	cannot determine swept geometry
KI_impossible_swing	55	cannot determine swung geometry
KI_impossible_taper	860	taper cannot be performed
KI_incompatible_curves	992	Curves not compatible for this operation
KI_incompatible_props	982	Combination of properties is impossible or ambiguous
KI_inconsistent_facesets	1020	failure to identify facesets
KI_inconsistent_geom	51	geometry is inconsistent with neighbors

KI_incorrect_mc_conf	1066	machine configuration not authorized for Parasolid
KI_instanced_body	1087	Body is instanced
KI_instanced_tools	541	instanced tool bodies
KI_insufficient_curves	985	Too many or too few curves
KI_insufficient_points	988	Insufficient control or spline points
KI_invalid_bodies	546	boolean failure or invalid bodies
KI_invalid_geometry	999	Geometry fails to pass checks
KI_is_attached	34	geometric entity is attached to topology
KI_journal_not_open	566	journal file not open
KI_key_in_use	530	key already used in archive
KI_key_not_found	58	key not found
KI_keyed_part_mismatch	913	archived part not same type as part in memory
KI_linear_multi_seg	1035	linear B-spline with >1 seg not allowed
KI_list_too_short	519	not enough items in list
KI_list_wrong_length	523	list not of expected length or is too long
KI_lt_3_sides	33	number of sides is less than 3
KI_majaxi_not_perpn	36	major axis and axis not perpendicular
KI_majrad_minrad_mismatch	507	incompatible values for major and minor radii
KI_mass_eq_0	82	bodies have zero total mass
KI_memory_full	901	modeler has run out of virtual memory
KI_mend_attempt_failure	1003	mending attempt has failed
KI_missing_geom	96	a topological entity lacks geometry
KI_mixed_sheets_solids	542	mixture of sheet and solid tool bodies
KI_modeller_not_started	931	STAMOD must be first call to KI
KI_modeller_not_stopped	932	calling STAMOD twice without calling STOMOD
KI_modified_sub_part	904	part has modified sub-part
KI_more_than_one_part	1090	More than one part in transmit file
KI_negative_body	106	body is inside out
KI_nitems_le_0	903	number of items is zero or negative
KI_nitems_lt_0	902	number of items is negative
KI_no_errors	0	Operation was successful
KI_no_eds_from_target	1036	no knitting pattern edges form target
KI_no_geometry	61	no geometry attached
KI_no_rollmark	854	no previous roll-mark exists
KI_no_user_fields	949	user fields are not in use
KI_non_manifold	547	non-manifold body or boundary
KI_non_orth_matrix	962	non-orthogonal matrix
KI_non_smooth_edge	1073	normals discontinuous across edge
KI_none_mergeable	48	no mergeable entities

KI_none_shared	1068	no shared geometry
KI_not_a_list	80	a list was not provided when one was required
KI_not_a_logical	908	value is not KI_true (1) or KI_false (0)
KI_not_a_tag	22	tag is invalid or dead
KI_not_at_rollmark	924	state of model changed since last roll-mark
KI_not_blended	335	no blend on edge
KI_not_connected	20	no such connection
KI_not_found	19	specified entity not found
KI_not_in_feat	512	entity not found in feature
KI_not_in_region	1077	not inside region
KI_not_in_same_body	1032	entities do not all belong to same the body
KI_not_in_same_part	59	entities do not all belong to the same part
KI_not_in_same_partition	1086	Entities are not all in the same partition
KI_not_in_same_shell	1070	entities not all from the same body
KI_not_interrupted	946	kernel has not been interrupted
KI_not_on_curve	67	not within resolution distance of curve
KI_not_on_edge	1013	not within resolution distance of edge
KI_not_on_face	54	not within resolution distance of face
KI_not_on_surface	915	not within resolution distance of surface
KI_not_same_length	359	lists should be the same length but are not
KI_not_sheet	939	body is not a sheet
KI_not_solid	101	a body is not solid
KI_not_unique	13	not unique
KI_null_arg_address	906	argument address given as zero
KI_null_axis	25	axis vector has zero length
KI_null_direction	31	direction vector has zero length
KI_opposed_sheets	951	attempt to unite opposed sheets
KI_order_lt_2	132	order is less than 2
KI_part_not_isolated	905	part not isolated
KI_part_not_keyed	524	part is not keyed and cannot be unloaded
KI_partial_coi_found	1040	failure due to detection of a partial coincidence
KI_partial_no_intersect	1067	no imprinting in local boolean
KI_pattern_invalid	1042	pattern would produce invalid body
KI_radii_both_0	3	both radii are zero
KI_radius_eq_0	925	radius is zero or very near to zero
KI_radius_le_0	16	radius is zero or negative
KI_radius_lt_0	18	radius is negative
KI_radius_sum_le_0	508	majrad < zero and majrad + minrad <= zero
KI_radius_too_large	926	radius is too large for modeler resolution
KI_receive_failed	63	receive failed

KI_recursive_call	934	calling KI before the previous call completes
KI_repeated_knots	983	Repeated knots in knot vector invalid in this context
KI_request_not_supported	997	Requested mass properties not implemented
KI_roll_forward_fail	856	roll forward not possible
KI_roll_is_off	855	logging for roll-back is disabled
KI_rollmark_failed	850	failed to set rollmark
KI_rot_angle_eq_0	32	zero angle rotation asked for
KI_run_time_error	947	non fatal run-time error
KI_same_tool_and_target	545	target body cannot also be a tool body
KI_save_snapshot_failed	562	failed to save snapshot
KI_sc_factor_le_0	40	scale must be greater than zero
KI_schema_access_error	970	open/close/read/write error for schema file
KI_schema_corrupt	971	contents of schema file not as expected
KI_sheet_untrimmed	1079	trim curves didn't remove any part of sheet
KI_size_mismatch	533	archived part has different size settings
KI_solid_body	1026	solid body is unsuitable for this operation
KI_solid_has_void	1084	solid body contains void
KI_still_referenced	68	argument is still referenced
KI_stol_too_small	960	invalid step length control set
KI_su_are_coincident	41	coincident surfaces
KI_su_self_intersect	141	would produce a self_intersecting surface
KI_system_error	900	modeler error: please report fault
KI_t_sheet	549	T-sheet
KI_tag_limit_exceeded	1049	tag limit would be exceeded
KI_tag_limit_out_of_range	1050	invalid tag limit
KI_targ_faces_many_bodies	1016	target faces in list are from more than one body
KI_tolerance_too_tight	1064	failed to meet tolerances
KI_too_many_control_pts	1001	parametric entity has too many control points
KI_too_many_derivatives	1010	too many derivatives requested
KI_too_many_targets	1054	too many target bodies
KI_tool_faces_many_bodies	1017	tool faces in list are from more than one body
KI_topol_not_from_body	1019	topology is not from expected body
KI_transmit_failed	560	transmit failed
KI_trim_loop_degenerate	1083	Trimming loop degenerates at given tolerance
KI_unchecked_entity	991	Entity unchecked and may be invalid for this operation
KI_unsuitable_entity	914	entity unsuitable for requested operation
KI_unsuitable_loop	1061	loop is of wrong type
KI_unsuitable_topology	1033	topology is unsuitable
KI_usfd_mismatch	936	file has wrong user-field size

KI_weight_le_0	111	weights are not all strictly positive
KI_wire_body	938	wire or acorn body is unsuitable for this operation
KI_withdrawn_surface	995	Part contains a withdrawn blend surface
KI_wrong_direction	961	start and end in wrong order
KI_wrong_entity	10	entity is not of type expected
KI_wrong_entity_in_list	357	entity of wrong type in list
KI_wrong_format	937	trying to open file in wrong format
KI_wrong_list_type	509	list is not of the expected type
KI_wrong_number_derivs	981	Too many or too few derivatives supplied
KI_wrong_number_edges	1028	only 2 edges at vertex permissible
KI_wrong_number_knots	980	Knot vector is too long or too short
KI_wrong_sub_type	553	sub-type of entity is unsuitable
KI_wrong_surface	950	wrong type of surface for operation
KI_wrong_transf	37	transformation is unsuitable for operation
KI_wrong_type_for_feat	513	entity type is inconsistent with feature type
KI_wrong_version	923	file is incompatible with this version

Kernel Interface Typedefs

C

C.1 Introduction

This appendix contains a list of all kernel interface typedefs and their meanings.

C.2 Alphabetical list of KI typedefs

C.2.1 KI_cod... typedefs

typedef	meaning	possible ifails
KI_cod_error	Error code (ifail) from the KI	
KI_cod_logical	Logical value KI_true or KI_false	KI_not_a_logical
KI_cod_ty	any type of entity <base>	KI_bad_type
KI_cod_tyen	entity type	KI_bad_type
KI_cod_tyge	geometry type	KI_bad_type
KI_cod_typt	point type	KI_bad_type
KI_cod_tycu	curve type	KI_bad_type
KI_cod_tysu	surface type	KI_bad_type
KI_cod_tybl	blending sub types	KI_bad_type
KI_cod_tyto	topology type	KI_bad_type
KI_cod_tyas	assembly type	KI_bad_type
KI_cod_tyin	instance type	KI_bad_type
KI_cod_tyad	assoc data type	KI_bad_type
KI_cod_tyat	attribute type	KI_bad_type
KI_cod_tyfe	feature type	KI_bad_type
KI_cod_tysa	system attribute type	KI_bad_type
KI_cod_tyua	user attribute type	KI_bad_type
KI_cod_tyli	list type	KI_bad_type
KI_cod_rq	any of request codes <base>	KI_bad_request_code
KI_cod_rqac	attribute class	KI_bad_request_code
KI_cod_rqap	attribute property	KI_bad_request_code
KI_cod_en	any of enquiry codes <base>	
KI_cod_enve	vertex property	
KI_cod_ened	edge property	
KI_cod_enlo	loop property	
KI_cod_ensh	shell property	

KI_cod_enby	body property	
KI_cod_enwr	wire property	
KI_cod_ense	sheet property	
KI_cod_enst	part state	KI_bad_spec_code
KI_cod_encl	enclosure	
KI_cod_slip	interface parameter	KI_bad_selection_code
KI_cod_slmp	modelling parameter	KI_bad_selection_code
KI_cod_slab	reason for abort	KI_bad_selection_code
KI_cod_sler	error enquiry	KI_bad_selection_code
KI_cod_slst	state enquiry	KI_bad_selection_code
KI_cod_sllo	local op. action	KI_bad_selection_code
KI_cod_rtlo	local op. return	
KI_cod_slfi	file enquiry	KI_bad_selection_code
KI_cod_slcp	control point size	KI_bad_selection_code
KI_cod_slba	parametric basis	KI_bad_selection_code
KI_cod_slle	simplification level	KI_bad_selection_code
KI_cod_slpk	pick return	KI_bad_type
KI_cod_rrop	rendering option	KI_bad_request_code
KI_cod_papr	parametric prop	KI_bad_parametric_prop
KI_cod_rtst	return state	
KI_cod_srop	standard rep opt	KI_bad_selection_code
KI_cod_maop	masspr option	KI_bad_selection_code
KI_cod_oufo	output format	KI_bad_selection_code
KI_cod_atop	attribute definition options	KI_bad_selection_code
KI_cod_mdop	mending option	KI_bad_selection_code
KI_cod_bbev	bulletin board event	KI_bad_selection_code
KI_cod_bbop	bulletin board option	KI_bad_selection_code
KI_cod_cicl	curve intersection classification	KI_bad_selection_code
KI_cod_clop	closest approach option	KI_bad_selection_code
KI_cod_cfcl	curve face classification	KI_bad_selection_code
KI_cod_imop	imprinting opt	KI_bad_selection_code
KI_cod_idop	identify region option	KI_bad_selection_code
KI_cod_rtto	CRTOBY returns	
KI_cod_byty	body types	KI_bad_selection_code
KI_cod_padi	parametric discontinuities	KI_bad_selection_code
KI_cod_rtcl	closest approach return	
KI_cod_rtmd	mending return	
KI_cod_mdfa	mending return	
KI_cod_rtkn	knitting return	
KI_cod_cbop	CRBYGE option	KI_bad_selection_code
KI_cod_blcc	first error from blending body	

KI_cod_blec	blend property	KI_bad_selection_code
KI_cod_inop	intersection option	KI_bad_selection_code
KI_cod_boop	boolean option	KI_bad_selection_code
KI_cod_crop	surface of revolution option	KI_bad_selection_code
KI_cod_sicl	surface intersect classification	KI_bad_selection_code
KI_cod_tsop	trimmed surface check option	KI_bad_selection_code
KI_cod_rtts	trimmed surface state code	
KI_cod_idty	common connection type	KI_bad_selection_code
KI_cod_chop	chcken option	KI_bad_selection_code
KI_cod_sltr	trimming options	
KI_cod_shop	sharen option	KI_bad_selection_code
KI_cod_loch	local checking level	KI_bad_selection_code
KI_cod_rtof	offset operations return	KI_bad_selection_code
KI_cod_pfop	enpifa option	KI_bad_selection_code
KI_cod_rttl	retlen status	
KI_cod_trsh	trshcu option	KI_bad_selection_code
KI_cod_fxft	blefxf option	KI_bad_selection_code
KI_cod_fxfe	blefxf error	KI_bad_selection_code

C.2.2 KI_chr... typedefs

typedef	meaning	possible ifails
KI_chr_key	key for archived part, etc.	KI_bad_key
KI_chr_name	name of entry	KI_bad_name
KI_chr_filename	filename eg. for a journal file	KI_bad_filename
KI_chr_string	string	KI_bad_string

C.2.3 KI_dbl... typedefs

typedef	meaning	possible ifails
KI_dbl	parameter range	
KI_dbl_angle	angle in radians	
KI_dbl_box	model space box	KI_bad_box
KI_dbl_coefficients	coefficients for parametric curve or surface	
KI_dbl_curvature	curvature	
KI_dbl_distance	distance	KI_distance_lt_0 or KI_distance_le_0
KI_dbl_knots	knots for parametric spline curve or surface	
KI_dbl_parameter	parametric curve or surface parameter	

KI_dbl_radius	radius	KI_radius_lt_0 or KI_radius_le_0
KI_dbl_sc_fact	scaling factor	KI_sc_factor_le_0
KI_dbl_tensor	tensor	
KI_dbl_transf_mx	transform matrix	
KI_dbl_view_mx	viewing matrix	

C.2.4 KI_int... typedefs

typedef	meaning	possible ifails
KI_int		
KI_int_bbitem	bulletin board item	
KI_int_dimension	dimension of vertices of parametric curve or surface	KI_bad_dimension
KI_int_id	entity identifier	
KI_int_index	index to a KI list or array	KI_bad_index
KI_int_nchars	length of a string	KI_bad_char_string
KI_int_nitems	number of items	KI_nitems_lt_0 or KI_nitems_le_0
KI_int_order	order of parametric curve or surface	KI_order_lt_2
KI_int_ufdval	user-field value	

C.2.5 KI_tag... typedefs

typedef	meaning	possible ifails
KI_tag	any tag	
KI_tag_assembly	tag of assembly entity	
KI_tag_attribute	tag of attribute entity	
KI_tag_attrib_def	tag of attribute type definition entity	
KI_tag_b_curve	tag of b_curve entity	
KI_tag_b_surface	tag of b_surface entity	
KI_tag_body	tag of body entity	
KI_tag_curve	tag of curve entity	
KI_tag_edge	tag of edge entity	
KI_tag_entity	tag of any entity	
KI_tag_face	tag of face entity	
KI_tag_feature	tag of feature entity	
KI_tag_geometry	tag of geometry entity	
KI_tag_instance	tag of instance entity	
KI_tag_list	tag of list entity	
KI_tag_loop	tag of loop entity	
KI_tag_paracurve	tag of paracurve entity	

KI_tag_parasurf	tag of parasurf entity	
KI_tag_part	tag of part entity	
KI_tag_point	tag of point entity	
KI_tag_shell	tag of shell entity	
KI_tag_sp_curve	tag of SP-curve entity	
KI_tag_surface	tag of surface entity	
KI_tag_topology	tag of topology entity	
KI_tag_transform	tag of transform entity	
KI_tag_vertex	tag of vertex entity	
KI_tag_list_dbl	tag of list of doubles	
KI_tag_list_int	tag of list of integers	
KI_tag_list_tag	tag of list of tags	
KI_tag_list_XXX	tag of list of tag of XXX entities	

C.2.6 KI_vec... typedefs

typedef	**meaning**	**possible ifails**
KI_vec_axis	axis direction vector	KI_null_axis
KI_vec_direction	general direction vector	KI_null_direction
KI_vec_normal	non zero surface normal	KI_null_direction
KI_vec_displacement	displacement vector	KI_null_direction
KI_vec_position	position within model space	KI_bad_position
KI_vec_derivatives	position and derivative vectors	
KI_vec_centre	centre of geometric entity	

Flick Function Descriptions

D

D.1 Introduction

This appendix describes all the FLICK functions currently available in KID. The names of all FLICK functions, both support functions and KI functions, are reserved words and should not be overwritten.

D.1.1 Documentation conventions

- basic data types - int, tag, double, vector, logical, char, string, address
- (data ...) - denotes a lisp list of type 'data'
- single element - denotes optional arguments
- < arguments . default > - denotes optional arguments with defaults
- argument | argument - denotes alternative possible values
- tokens should be quoted, e.g. 'tytofa

D.2 Support functions

alloc - KI support function to create workspace

Syntax:	(alloc int) => address
Args:	count
Returns:	workspace
Notes:	The count is the number of integers the workspace will store. The workspace returned is initialized to zeros.

allow_ifails - KI support function to control ifail error handling

Syntax:	(allow_ifails (< string > token ...) expression)
Args:	(KI_name valid_ifail ...) lisp_programme
Returns:	expression_value

Initially the only valid ifail is zero (KI_no_errors), any other ifail returned will generate a lisp error. The set of valid ifails can be extended by including those which are to be allowed in the valid ifail specification either for all KI calls or if a KI function name is given as the first element of the valid ifail list only for the named KI function. Calls to allow_ifails can be nested. Once an ifail has been made valid it cannot subsequently be disallowed.

To allow all ifails the (pseudo) token KI_all_ifails is provided.

To indicate all but the specified ifails use a negative sign (e.g. - KI_not_a_tag).

> **Note:** Any KI call which returns an ifail will return all its other arguments as nulls and zeros.

Examples

```
( allow_ifails ( KI_missing_geom ) ( IDSOFF 99 ))
--- call IDSOFF but don't produce an error if the face lacks
geometry
( allow_ifails ( STOMOD KI_modeller_not_started ) ( STOMOD ))
--- stop the modeller but don't complain if it isn't started
( allow_ifails ( KI_roll_is_off ) ( my_strict_programme ))
--- allow KI_roll_is_off errors, but no others
( allow_ifails ( - KI_corrupt_file ) ( my_liberal_programme ))
--- allow all errors except KI_corrupt_file
( allow_ifails ( KI_all_ifails ) ( my_careless_programme ))
--- don't complain about anything
```

For greater flexibility of use the valid ifail specification may be a list of valid ifail specifications, as in the following example:

```
( allow_ifails ( ( IDSOFF KI_missing_geom KI_not_a_tag )
    ( IDCOEN KI_missing_geom KI_not_a_tag )
    ( setq v1 ( IDSOFF 99 ) )
    ( setq v2 (IDCOEN 100 ) ) )
```

This function does not evaluate its arguments, which is more convenient for normal use but means that it is difficult to pass arguments to it inside another function. The support function **valid_ifails** does evaluate its arguments and is intended for use inside user functions rather than directly.

empty - KI support function to turn an array into a lisp list

Syntax:	(empty int type address) => (values ...)
Args:	count data_type array
Returns:	(data ...)
Example:	(empty 6 tag @12345)
Notes:	This function unpacks arrays and generates lisp lists. The supported types are: logical, char, int, double, string, vector, tag.
	If 0 values are requested the returned list is always nil. The inverse function to ‘empty’ is ‘fill’.

enlist - KI support function to turn the lisp lists into KI lists

Syntax:	(enlist values <token>) => tag
Args:	data < type . TYLITG >
Returns:	KI_list
KI calls:	CRLIST, PTTGLI, PTINLI, PTRLLI
Example:	(enlist '(3.0 4 5.3) 'TYLIRL) (enlist '(7 8 9)) (enlist 7)

Notes:	This function turns the given values into a sensible KI list. The list type is an optional argument which defaults to TYLITG. The empty list generates the null tag (0). Atomic data is treated as a list of one element. Tag lists with a single element are not converted (since the KI will accept these as-is). The inverse function to 'enlist' is 'unlist'.

fill - KI support function to turn lists into arrays

Syntax:	(fill data_type '(values ...)) => address
Args:	data_type data_list
Returns:	array_address
Example:	(fill vector '((0 0 0) (0 0 1)))
Notes:	The array created may be passed to the KI, its value is an address. The possible data types are: logical, char, int, double, string, vector, tag. When given an empty list the function produces a valid pointer to a zero word. This is because the KI will not accept a null pointer (@0) as a valid argument. The inverse function to 'fill' is 'empty'.

ifails - KI support function to map ifails to ints

Syntax:	(ifails int) => string (ifails string) => int
Args:	value
Returns:	corresponding_value
Example:	(ifails 'KI_run_time_error) (ifails 947) (ifails '(947 948))
Notes:	When applied to integers this returns the equivalent symbolic ifail. When applied to symbols it returns the equivalent integer. When applied to lists it acts on each element. When applied to invalid integers it gives the value UNKNOWN_IFAIL. When applied to invalid symbols it gives the value -1, and in this case the KI itself will raise an error.

timing - controls output of timing data for KI function calls

Syntax:	(timing <0|1|2|t|nil>) => int
Args:	(timing <level>)
Returns:	current_level

token - KI support function to map tokens to ints

Syntax:	(token int) => string (token string) => int
Args:	value
Returns:	corresponding_value
Example:	(token 'RTLOSX) (token 15301) (token '(TYTOFA TYGESU))
Notes:	When applied to integers this returns the equivalent symbolic token. When applied to symbols it returns the equivalent integer. When applied to lists it acts on each element. When applied to invalid integers it gives the value UNKNOWN_TOKEN. When applied to invalid symbols it gives the value -1, and in this case the KI itself will raise an error.

unlist - KI support function to turn KI lists into lisp lists

Syntax:	(unlist tag <int> <token>)
Args:	list <length . nil> <list_type . TYLITG>
Returns:	(data ...)
KI calls:	GTRLLI, GTTGLI, GTINLI, COLIST, DELENT
Example:	(unlist 77 5 'TYLIRL) (unlist 88 nil 'TYLIIN) (unlist 8)
Notes:	The list length and list type are optional arguments. If the length is supplied as nil the whole list is returned. The default length is nil, the default list type is TYLITG. The null tag (0) returns the nil list regardless. ‘unlist’ deletes the list after unpacking it using DELENT. The inverse function to ‘unlist’ is ‘enlist’.

valid_ifails - suppress lisp error generation from Parasolid errors

Syntax:	(valid_ifails list expression)
Args:	ifail_spec code_to_evaluate
Returns:	result of evaluating expression

Notes:	ifail spec='(<KI_function_name><->KI_ifail_name<->KI_ifail_name ...) or '(ifail_spec ifail_spec ...)
	The presence of KI_ifail_name in the list suppresses errors from that ifail, unless it is preceded by '-' when all ifails but that ifail will be suppressed. The presence of a KI_function_name at the head of the list causes the rest of the list to be applied to calls to that function only.
	The special ifail code KI_all_ifails is provide to suppress or unsuppress all ifails.
	This function is primarily intended for use inside other functions. The fact that it evaluates its arguments means that for direct use they must normally be quoted. The function allow_ifails does not evaluate its arguments and is the function which should normally be used directly

D.3 KI Functions

ADPAPE - KI function to add parameter (line) to B-curve or B-surface

Syntax	(ADPAPE tag double int) => (ifail)
Args	entity parameter uorv
Returns	(ifail)
Syntax	(adpape tag double token) => t
Args	entity parameter uorv
Returns	t
Example	(adpape 20 0.3 'PAPRUP)
Note	The geometry must not be attached to any topology.

ADVXED - KI function to add a new vertex to a given edge

Syntax	(ADVXED tag vector) => (tag tag ifail)
Args	edge point
Returns	(newvrx newedg ifail)
Syntax	(advxed tag vector) => (tag tag)
Args	edge point
Returns	(newvrx newedg)
Example	(advxed (e1 tag) '(2.5 0 5))

APPTRA - KI function to apply a transformation

Syntax	(APPTRA tag tag) => (ifail)
Args	entity transf
Returns	(ifail)
Syntax	(apptra tag tag) => tag
Args	entity transf
Returns	entity
Example	(apptra (b0 tag) 425) (apptra (list (b0 tag) (c0 tag)) 425)
Note	Mirroring and scaling transformations cannot be applied to assemblies or instances.

ATGETO - KI function to attach geometry to topology

Syntax	(ATGETO int address address int address) => (ifail)
Args	ntopol topology senses ngeom geometry
Returns	(ifail)
Syntax	(atgeto (tag ...) <(logical ...)> (tag ...)) => t
Args	(topology ...) <(senses ...)> (geometry ...)
Returns	t

ATTGEO - KI function to attach geometry to topology

Syntax	(ATTGEO tag tag logical) => (ifail)
Args	topol geom sense
Returns	(ifail)
Syntax	(attgeo tag tag logical) => t
Args	topol geom sense
Returns	t

BLECHK - KI function to check the local validity of unfixed blends

Syntax	(BLECHK tag int) => (int tag tag tag ifail)
Args	edge level
Returns	(n_invalid code_list bad_edge_list tag_list ifail)
Syntax	(blechk (tag ...) int) => ((token ...) (tag ...) (tag ...))

Args	'(edge ...) < level . 1 >
Returns	((error edge topol) ...)
Example	(blechk '(22 34 45) 2) (blechk (list (e1 tag) (e2 tag) (e3 tag)) 2)

BLECRB - KI function to define a rolling ball blend or chamfer

Syntax	(BLECRB tag int double double int address address) => (tag int ifail)
Args	edge type range1 range2 nprops prop_array tag_array
Returns	(affected_edges n_edges ifail)
Syntax	(blecrb tag int (double <double>) ((token value) ...))
Args	'(edge ...) type (range1 <range2>) (property value) ...
Returns	(affected_edge ...)
Example	(blecrb '(20 22 24) 1 1.75 '((BLECCL 42) (BLECPR))) (blecrb (list (e1 tag) (e2 tag) (e3 tag) 1 1.75 '((BLECCL 42) (BLECPR)))

BLECVR - KI function to define a variable radius blend

Syntax	(BLECVR tag int address address int address address) => (tag int ifail)
Args	edge n_points vector_arr real_arr n_props prop_arr tag_arr
Returns	(affected_edges n_edges ifail)
Syntax	(blecvr tag (vector ...) ((double <double>) ...) ((token value) ...))
Args	edge '(position ...) '((lradius rradius) ...) '((property value) ...)
Returns	(affected_edge ...)
	If only one radius is supplied at each point for both left and right sides this need not be put into a sublist.
Example	(blecvr '(20) '((0 0 1) (0 0 5)) '(1.75 3.5) '((BLECTL 0.001) (BLECLI) (BLECPR)))

BLEENQ - KI function to enquire blend parameters

Syntax	(BLEENQ tag) => (tag tag int double double int tag tag ifail)
Args	edge
Returns	(face1 face2 type_code range1 range2 n_props props values ifail)
Syntax	(bleenq tag) => (tag tag (int ...) (real ...))
Args	edge
Returns	(face1 face2 type range1 range2 (property value ...) ...)
Example	(bleenq 56)

BLEFIX - KI function to fix blends in a body

Syntax	(BLEFIX tag) => (int tag tag int tag tag ifail)
Args	body
Returns	(n_blend_faces blend_faces underlying_faces error edge topol ifail)
Syntax	(blefix tag) => ((tag ...) (tag ...))
Args	body
Returns	(error edge topol (blend_face (underlying_face ...)) ...)
Example	(blefix (b0 tag))
Note	If no errors are found then KI_FALSE will be returned.

BLEFXF - KI function to create a blend between specified faces

Syntax	(BLEFXF tag tag logical logical int address address) => (int tag int tag tag ifail)
Args	left_wall right_wall left_rev right_rev nopts opts opt_data
Returns	(status status_data nblends blends underlying_data ifail)
Syntax	(blefxf tag tag logical logical <'((token tag|double ...) ...)>) => (token (<tag ...>) (tag (tag ...)) <...>)
Args	'(left_face ...) '(right_face ...) left_rev right_rev <'((option data) ...)>
Returns	(status (<data ...>) (blend (underlying ...)) <...>)
Example	(blefxf 80 75 t t '(FXFTCB 0.5))

BLEREM - KI function to remove blend attributes

Syntax	(BLEREM tag) => (ifail)
Args	edges
Returns	(ifail)
Syntax	(blerem (tag ...)) => t
Args	'(edge ...)
Returns	t
Example	(blerem '(20 22 24))

BLNAFF - KI function to find edges and faces affected by blend

Syntax	(BLNAFF tag) => (tag int tag int ifail)
Args	edge

Returns	(edge_list length face_list length ifail)
Syntax	(blnaff tag) => ((tag ...) (tag ...))
Args	edge
Returns	((edge ...) (face ...))

BLNDVX - KI function to blend vertices on sheets and wires

Syntax	(BLNDVX tag double) => (tag tag ifail)
Args	vertices radius
Returns	(new_edges new_vertices ifail)
Syntax	(blndvx tag double) => ((tag ...) (tag ...))
Args	vertices radius
Returns	((new_edge ...) (new_vertex ...))
Example	(blndvx 19 3.2) (blndvx '(19 21 23) 1.4)

BOPBYS - KI function to do a global or local boolean operation on bodies

Syntax	(BOPBYS tag tag int address address) => (tag int ifail)
Args	targets tools n_opts options_arr data_arr
Returns	(bodies n_body ifail)
Syntax	(bopbys (tag ...) (tag ...) ((token <tag> ...) ...)) => (tag ...)
Args	targets tools '((option <topol> ...) ...)
Returns	(body ...)
Example	(bopbys 20 '(22 24 26) '((BOOPSU) (BOOPME))) (bopbys (b0 tag) (list (b1 tag) (b2 tag) (b3 tag)) '((BOOPSU) (BOOPME)))

CCLIST - KI function to concatenate two lists, tail onto head

Syntax	(CCLIST tag tag) => (ifail)
Args	head tail
Returns	(ifail)
Syntax	(cclist tag tag) => tag
Args	head tail
Returns	head

CHCKEN - KI function to check an entity

Syntax	(CHCKEN tag int int address) => (tag tag tag int ifail)
Args	entity max_faults n_opts opt_arr
Returns	(list_of_tokens list_of_tags list_of_tags int ifail)
Syntax	(chcken tag <int> (token ...)) => ((token <tag> <vector>) ...)
Args	entity <n_errors> '(<options> ...)
Returns	((error <entity> <position>) ...)
Note	If no errors are found then KI_no_errors will be returned.

CLABYS - KI function to detect clashing bodies

Syntax	(CLABYS tag tag tag tag logical) => (tag tag int ifail)
Args	body transform body transform full
Returns	(face_list face_list n_clashes ifail)
Syntax	(clabys tag tag tag tag logical) => ((tag ...) (tag ...))
Args	body transform body transform full
Returns	((face ...) (face ...))
Example	(clabys 125 nil 145 nil t)
Note	Whereas CLABYS may return two lists of faces or just two faces, clabys always returns two (LISP) lists of faces. If the bodies do not clash the function returns nil.

CLENEN - KI function to find the closest point between two entities/entity lists

Syntax	(CLENEN tag tag int address address) => (double tag tag vector vector address address tag int tag int ifail)
Args	entity_list entity_list nopts optlist optdata
Returns	(distance entity_list entity_list point1 point2 params1 params2 prop1 nprop1 prop2 nprop2 ifail)
Syntax	(clenen tag tag <((token double ...)) ...)>) => (double (tag ...) (tag ...) vector vector (double double) (double double) (token ...) (token ...))
Args	'(entity ...) '(entity ...) < optlist >
Returns	(distance (entity sub_topol) (entity sub_topol) point1 point2 (param1 param1) (param2 param2) (token ...) (token ...))

Example	(clenen 21 79) (clenen '(21 32) '(34 45) '((CLOPP1 2.0 2.0)))
Note	The parametrization will depend upon the type of sub_topology but, with clenen, it will always be decoded as two doubles. The KI manual should be consulted to determine which parameters are meaningful for given sub_topologies.

CLENEX - KI function to find the closest point between entities

Syntax	(CLENEX tag tag int address address) => (int tag tag tag tag tag tag tag tag tag tag tag ifail)
Args	entity1_list entity2_list nopts optlist optdata
Returns	(num_minima min_distances topol1_lists topol2_lists point1_list point2_list parms1_lists parms2_lists prop1_lists nprop1_list prop2_lists nprop2_list ifail)
Syntax	(clenex tag tag <((token double ...) ...)>) => (double ... ((tag ...) ...) ((tag ...) ...) (vector ...) (vector ...) ((double double) ...) ((double double) ...) ((<token> ...) ...) ((<token> ...) ...))
Args	'(entity ...) '(entity ...) < optlist >
Returns	((distance ...) ((entity1 sub_topol) ...) ((entity2 sub_topol) ...) (point1 ...) (point2 ...) ((param11 param12) ...) ((param21 param22) ...) ((<token1> ...) ...) ((token2 ...) ...))
Example	(clenex 21 79) (clenex 21 79 '(CLOPFA)) (clenex '(21 32) '(34 45) '((CLOPP1 2.0 2.0)))
Note	The parametrization will depend upon the type of sub_topology but, with clenex, it will always be decoded as two doubles. The KI manual should be consulted to determine which parameters are meaningful for given sub_topologies.

CLPTEN - KI function to find the closest point on an entity to a given point

Syntax	(CLPTEN vector tag int address address) => (double tag vector address tag int ifail)
Args	point entity_list nopts opt_list opt_data
Returns	(min_dist topol_list soln_pt parms props nprops ifail)
Syntax	(clpten vector tag <((token (double ...)) ...)>) => (double (tag ...) vector (<double> <double>) (<token> ...))
Args	point entity_list < optlist >

Returns	(min_dist topol_list soln_pt parms prop_list)
Example	(clpten '(3 3 8) (b0 tag) '((CLOPUP 4) (CLOPTL 0.1)))

CLPTEX - KI function to find the closest point(s) on an entity to a given point

Syntax	(CLPTEX vector tag int address address) => (int tag tag tag tag tag tag ifail)
Args	point entity_list nopts opt_list opt_data
Returns	(num_minima min_distance topol_lists soln_points parms_lists prop_lists nprops_list ifail)
Syntax	(clptex vector tag <((token (double ...) ...)>) => ((double ...) ((tag ...) ...) (vector ...) ((double double) ...) ((<token> ...) ...))
Args	point entity_list < optlist >
Returns	(min_distances topol_lists soln_points parms_lists prop_lists)
Example	(clptex '(3 3 8) (b0 tag) '((CLOPUP 4) (CLOPTL 0.1)))

CLPTFA - KI function to find the closest point on a face to a given point

Syntax	(CLPTFA vector tag int address address) => (vector address tag logical ifail)
Args	point face nopts optlist optdata
Returns	(fpoint params topol ortho ifail)
Syntax	(clptfa vector tag <((token double ...) ...)>) => (vector (double double) tag logical)
Args	point face < optlist >
Returns	(fpoint svec topol ortho)
Example	(clptfa '(2 2 2) (f0 tag)) (clptfa '(2 2 2) (f0 tag) '(CLOPPR 2.0 2.0))

COFEAT - KI function to count the entities in a feature

Syntax	(COFEAT tag) => (int ifail)
Args	feature
Returns	(n_items ifail)
Syntax	(cofeat tag) => int
Args	feature
Returns	n_items

COLIST - KI function to count the items in a list

Syntax	(COLIST tag) => (int ifail)
Args	list
Returns	(n_items ifail)
Syntax	(colist tag) => int
Args	list
Returns	n_items

COMENT - KI function to comment the journal file

Syntax	(COMENT int string) => (ifail)
Args	nchars comment
Returns	(ifail)
Syntax	(coment string) => t
Args	comment
Returns	t
Example	(coment “end of test”)

COPYEN - KI function to copy an entity

Syntax	(COPYEN tag) => (tag ifail)
Args	entity
Returns	(new_entity ifail)
Syntax	(copyen tag) => tag
Args	entity
Returns	new_entity

CRATDF - KI function to create a new attribute type definition

Syntax	(CRATDF int string int address address) => (tag ifail)
Args	namlen name nopts opt_array opt_data
Returns	(att_type ifail)
Syntax	(cratdf string token '(token ...) <'(token ...)>) => tag
Args	name class owners <fields>
Returns	att_type
Example	(cratdf 'Mass 'RQAC02 '(TYTOAS TYTOIN TYTOBY))

CRBSPC - KI function to create a B-curve from B-spline data

Syntax	(CRBSPC int int int address address int address) => (tag ifail)
Args	dim order ncontrol controls knots nprops properties
Returns	(B-curve ifail)
Syntax	(crbspc int int '(real ...) '(real ...) '(token)) => tag
Args	dim order '(control...) '(knot ...) '(property)
Returns	B-curve
Example	(crbspc 3 4 '((0 0 0) (0 0 1) (0 1 2)) '(0 1 1.5 3) '(PAPRPE))
Note	The list of control points is flattened before use, so any embedded list or vector bracketing may be used.

CRBSPS - KI function to create a B-surface from B-spline data

Syntax	(CRBSPS int int int int int address address address int address) => (tag ifail)
Args	dim uord vord ncol nrow controls uknots vknots nprops props
Returns	(B-surface ifail)
Syntax	(crbsps int int int int int '((double ...) ...) '(double ...) '(double ...) '(token ...)) => tag
Args	dim uord vord ncol nrow '(control ...)'(uknot ...) '(vknot ...) '(property ...)
Returns	B-surface
Example	(crbsps 3 3 3 6 6 '((0 0 0) (0 0 1) ......) '(0 0.2 0.5 0.8 1 1.2 1.5 1.8 2) '(0 0.2 0.5 0.8 1 1.2 1.5 1.8 2) nil)
Note	The controls points are flattened before use and so any embedding of brackets may be used.

CRBXSO - KI function to create a box solid (cuboid)

Syntax	(CRBXSO vector vector double double double) => (tag ifail)
Args	centre axis width length height
Returns	(body ifail)
Syntax	(crbxso vector vector double double double) => tag
Args	centre axis width length height
Returns	body
Example	(crbxso '(0 0 0) '(0 0 1) 5 5 5)

CRBYGE - KI function to create a body from geometry

Syntax	(CRBYGE tag int address address) => (tag ifail)
Args	geometry nopts opt_list opt_data
Returns	(wire_or_sheet ifail)
Syntax	(crbyge tag <'((token double double) <...>)>) => tag
Args	geometry <'((option start end) <(option start end)>))>
Returns	wire_or_sheet
Example	(crbyge 17) (crbyge 17 '(CBOPUR 0.0 0.1)) (crbyge 17 '((CBOPUR 0.0 1.0) (CBOPVR 0.0 3.0)))

CRCAPO - KI function to create a Cartesian point

Syntax	(CRCAPO double double double) => (tag ifail)
Args	x y z
Returns	(point ifail)
Syntax	(crcapo vector) => tag
Args	coords
Returns	point

CRCICU - KI function to create a circular curve

Syntax	(CRCICU vector vector double) => (tag ifail)
Args	centre axis radius
Returns	(circle ifail)
Syntax	(crcicu vector vector double) => tag
Args	centre axis radius
Returns	circle
Example	(crcicu '(0 0 0) '(0 1 0) 0.5)

CRCMPC - KI function to join B-curves into a single curve

Syntax	(CRCMPC int address) => (tag ifail)
Args	ncurves curve_array
Returns	(B-curve ifail)
Syntax	(crcmpc '(tag ...)) => tag

Args	'(B-curve ...)
Returns	B-curve

CRCOSO - KI function to create a conical solid

Syntax	(CRCOSO vector vector double double double) => (tag ifail)
Args	centre axis base_rad top_rad height
Returns	(cone ifail)
Syntax	(crcoso vector vector double double double) => tag
Args	centre axis base_rad top_rad height
Returns	cone

CRCOSU - KI routine to create a conical surface

Syntax	(CRCOSU vector vector double double) => (tag ifail)
Args	position axis_direction radius_at_position half-angle
Returns	(cone ifail)
Syntax	(crcosu vector vector double double) => tag
Args	position axis_direction radius_at_position half-angle
Returns	cone

CRCPCU - KI function to create a constant parameter line curve on a surface

Syntax	(CRCPCU tag int double) => (tag ifail)
Args	surface u_or_v param
Returns	(curve ifail)
Syntax	(crcpcu tag token param) => tag
Args	surface u_or_v param
Returns	curve
Example	(crcpcu 17 'PAPRUP 0.1)

CRCUPC - KI function to create a B-curve from a general curve

Syntax	(CRCUPC tag address) => (tag int ifail)
Args	curve bounds_array
Returns	(B-curve nseg ifail)

Syntax	(crcupc tag vector vector) => (tag int)
Args	curve start end
Returns	(B-curve nseg)

CRCYSO - KI function to create cylindrical solid

Syntax	(CRCYSO vector vector double double) => (tag ifail)
Args	centre axis radius height
Returns	(body ifail)
Syntax	(crcyso vector vector double double) => tag
Args	centre axis radius height
Returns	body

CRCYSU - KI function to create cylindrical surface

Syntax	(CRCYSU vector vector double) => (tag ifail)
Args	position axis radius
Returns	(surface ifail)
Syntax	(crcysu vector vector double) => tag
Args	position axis radius
Returns	surface

CREASS - KI function to create assembly

Syntax	(CREASS int) => (tag ifail)
Args	type
Returns	(assembly ifail)
Syntax	(creass) => tag
Args	-none-
Returns	assembly
Note	There is only one type of assembly and this is defaulted in creass.

CREATT - KI routine to create an attribute

Syntax	(CREATT tag tag int address int address int address int string) => (ifail)
Args	owners att_type nint int_array nreal real_array nstring int_array nchars string
Returns	(ifail)

Syntax	(creatt '(tag ...) tag '(int ...) '(double ...) '(string ...)) => t
Args	owners att_type int_list double_list string_list
Returns	t
Example	(creatt 17 133 nil '(1.0) '(density)) (creatt '(17 23) 133 '(1 1 1) nil nil)

CREFEA - KI function to create a feature

Syntax	(CREFEA int tag) => (tag ifail)
Args	type owner
Returns	(feature ifail)
Syntax	(crefea token tag) => tag
Args	type owner
Returns	feature

CREINS - KI function to create an instance of a part within an assembly

Syntax	(CREINS tag tag tag int) => (tag ifail)
Args	assembly part transform type
Returns	(instance ifail)
Syntax	(creins tag tag <tag>) => tag
Args	assembly part <transform>
Returns	instance
Example	(creins 99 7)
Note	If a transform is not supplied it defaults to the null tag (0) which is interpreted as the identity transform. Also note that there is only one type of instance (TYINPS) and this is always used by creins.

CRELCU - KI function to create an elliptic curve

Syntax	(CRELCU vector vector double vector double) => (tag ifail)
Args	centre axis major_radius major_axis minor_radius
Returns	(ellipse ifail)
Syntax	(crelcu vector vector double vector double) => tag
Args	centre axis major_radius major_axis minor_radius
Returns	ellipse

CREQSC - KI function to create an equal scaling transformation

Syntax	(CREQSC double vector) => (tag ifail)
Args	scale centre
Returns	(transform ifail)
Syntax	(creqsc double vector) => tag
Args	scale centre
Returns	transform

CREREF - KI function to create a reflection transform

Syntax	(CREREF vector vector) => (tag ifail)
Args	position normal
Returns	(transform ifail)
Syntax	(creref vector vector) => tag
Args	position normal
Returns	transform

CREROT - KI function to create a rotation transformation

Syntax	(CREROT vector vector double) => (tag ifail)
Args	position axis angle
Returns	(transform ifail)
Syntax	(crerot vector vector double) => tag
Args	position axis angle
Returns	transform

CRETFM - KI function to create a general transformation from a given matrix

Syntax	(CRETFM address) => (tag ifail)
Args	coefficient_array
Returns	(transform ifail)
Syntax	(cretfm '(double ...)) => tag
Args	'(transform_coefficients ...)
Returns	transform
Note	16 coefficients must be supplied to define the transform.

CRETRA - KI function to create a translation transformation

Syntax	(CRETRA vector double) => (tag ifail)
Args	direction distance
Returns	(transformation ifail)
Syntax	(cretra vector double) => tag
Args	direction distance
Returns	transform
Example	(cretra '(0 0 0) 2)

CREXSU - KI function to create an extruded surface

Syntax	(CREXSU tag vector logical) => (tag ifail)
Args	curve path simplify
Returns	(surface ifail)
Syntax	(crexsu tag vector <logical>) => tag
Args	curve path <simplify . nil>
Returns	surface

CRFASU - KI function to create a surface to fit and attach to face

Syntax	(CRFASU tag) => (int tag int ifail)
Args	face
Returns	(sf_type surface body_state ifail)
Syntax	(crfasu tag) => (token tag token)
Args	face
Returns	(sf_type surface body_state)

CRFGCU - KI function to create a foreign geometry curve

Syntax	(CRFGCU int string int int address int address) => (tag ifail)
Args	keylen key nspace nints ivals nreals rvals
Returns	(curve ifail)
Syntax	(crfgcu string <int> <'(int ...)> <'(double ...)> => tag
Args	key <nspace> <'(ival ...)> <'(rval ...)>

Returns	curve
Example	(crfgcu “SDL/helix”) (crfgcu “SDL/sine” 1 () '(1.0 2.0))

CRFGSU - KI function to create a foreign geometry surface

Syntax	(CRFGSU int string int int address int address) => (tag ifail)
Args	keylen key nspace nints ivals nreals rvals
Returns	(surface ifail)
Syntax	(crfgsu string <int> <'(int ...)> <'(double ...)> => tag
Args	key <nspace> <'(ival ...)> <'(rval ...)>
Returns	surface
Example	(crfgsu “SDL/franke”) (crfgsu “SDL/corrugated” 1 '(1.0 2.0 3.0))

CRINCU - KI function to create intersection curves

Syntax	(CRINCU tag tag address) => (tag int ifail)
Args	surface surface box_array
Returns	(curve_list n_curves ifail)
Syntax	(crincu tag tag <((double...)...)>) => (tag ...)
Args	surface surface <box.((-500 -500 -500) (500 500 500))>
Returns	(curve ...)
Example	(crincu 17 18) (crincu 17 18 '((-10 -10 -10) (10 10 10))) (crincu 17 18 '((-10 10) (-10 10) (-10 10)))
Note	The KI box is an array of doubles with all low coords first. The lisp box can be given as two extreme vectors or as 3 intervals. The box argument is optional in crincu. The default is for the whole of the standard size box.

CRKNPA - KI function to create a knitting pattern from a list of bodies

Syntax	(CRKNPA tag) => (tag tag int tag int tag int ifail)
Args	bodies
Returns	(edges1 edges2 nedges negated n_negated over n_over ifail)
Syntax	(crknpa '(tag ...)) => ((<tag> ...) (<tag> ...) (<tag> ...) (<tag> ...))
Args	'(body ...)

Returns	((<edge> ...) (<edge> ...) (<body> ...) (<body> ...))
Example	(crknpa '(17 91))
Note	The bodies passed to the function must be either solid or sheet bodies. Any leftover bodies have no edges in the pattern.

CRLFPS - KI function to create a B-surface by lofting

Syntax	(CRLFPS int address int address address 0 => (tag ifail)
Args	ncurves curve_array nprops prop_array tag_array
Returns	(B-surface ifail)
Syntax	(crlfps '(tag ...) '(token <real> ...) ...) => tag
Args	'(curve ...) '(prop <data>) ...
Returns	B-surface
Example	(crlfps '(18 95 43 50) '(PAPRPE) '(PAPRIS 1))
Note	crlfps has a variable number of arguments consisting of several lists, each containing a token and (where appropriate) associated real data.

CRLICU - KI function to create a linear curve

Syntax	(CRLICU vector vector) => (tag ifail)
Args	position direction
Returns	(line ifail)
Syntax	(crlicu vector vector) => tag
Args	position direction
Returns	line
Note	Creates a line through the given point in the given direction.

CRLIST - KI function to create an (empty) list entity

Syntax	(CRLIST int) => (tag ifail)
Args	type
Returns	(list ifail)
Syntax	(crlist token) => tag
Args	type
Returns	list
Example	(crlist 'TYLITG)

Note	The support function enlist performs convenient conversion between KI lists and lisp lists.

CRMINO - KI function to create a minimum object

Syntax	(CRMINO) => (tag ifail)
Args	- none -
Returns	(body ifail)
Syntax	(crmino) => tag
Args	- none -
Returns	body

CROFSU - KI function to create an offset surface

Syntax	(CROFSU tag double) => (tag ifail)
Args	underlying_surface distance
Returns	(surface ifail)
Syntax	(crofsu tag double) => tag
Args	underlying_surface distance
Returns	surface
Example	(crofsu 84 12.0)

CRPLSU - KI function to create a planar surface

Syntax	(CRPLSU vector vector) => (tag ifail)
Args	position normal
Returns	(surface ifail)
Syntax	(crplsu vector vector) => tag
Args	position normal
Returns	surface

CRPRSO - KI function to create a prismatic solid

Syntax	(CRPRSO vector vector double int double) => (tag ifail)
Args	centre axis radius nsides height
Returns	(body ifail)
Syntax	(crprso vector vector double int double) => tag

Args	centre axis radius nsides height
Returns	body

CRPWPC - KI function to create a B-curve from piecewise data

Syntax	(CRPWPC int int int address int) => (tag ifail)
Args	dim ord nseg coeff_array basis
Returns	(B-curve ifail)
Syntax	(crpwpc int int int '(vector ...) token) => tag
Args	dim ord nseg '(point ..) basis
Returns	B-curve

CRPWPS - KI function to create a B-surface from piecewise data

Syntax	(CRPWPS int int int int int address int) => (tag ifail)
Args	dim uord vord ncol nrow coeff_array basis
Returns	(B-surface ifail)
Syntax	(crpwps int int int int int '(vector ...) token) => tag
Args	dim uord vord ncol nrow '(point ...) basis
Returns	B-surface

CRRVSU - KI function to create a surface of revolution

Syntax	(CRRVSU tag vector vector int address address) => (tag ifail)
Args	curve point axis n_opts options real_lists
Returns	(surface ifail)
Syntax	(crrvsu tag vector vector '((token <real> <real>) ...)) => tag
Args	curve point axis '((option data ...) ...)
Returns	surface

CRSEPS - KI function to sweep a B-curve into a B-surface

Syntax	(CRSEPS tag vector) => (tag ifail)
Args	B-curve translation
Returns	(B-surface ifail)
Syntax	(crseps tag vector) => tag

Args	B-curve translation
Returns	B-surface

CRSHFA - KI function to create a sheet body from a face

Syntax	(CRSHFA tag) => (tag ifail)
Args	face
Returns	(body ifail)
Syntax	(crshfa tag) => tag
Args	face
Returns	body

CRSIPS - KI function to swing a B-curve into a B-surface

Syntax	(CRSIPS tag vector vector double) => (tag ifail)
Args	B-curve point axis angle
Returns	(B-surface ifail)
Syntax	(crsips tag vector vector double) => tag
Args	B-curve point axis angle
Returns	B-surface

CRSOFA - KI function to create solid from faces

Syntax	(CRSOFA tag int) => (tag int tag ifail)
Args	face_list action
Returns	(body_list nbodies state_list ifail)
Syntax	(crsofa '(tag ...) token) => ((tag token) ...)
Args	'(face ...) action
Returns	((body state) ...)
Note	A face or a list of faces may be supplied to crsofa.

CRSPCU - KI function to create SP-curve(s) from B-spline data defined in surface parameter space

Syntax	(CRSPCU tag int int int address address logical logical) => (int tag ifail)
Args	surf dim order nctrl ctrls knots periodic split
Returns	(n_curves SP-curve ifail)

Syntax	(crspcu tag int int '((double ...) ...) '(double ...) logical logical) => (tag ...)
Args	surf dim order '(ctrl_pt ...) '(knot ...) <periodic . f> <split . f>
Returns	(SP-curve ...)
Example	(crspcu 17 3 3 '((0.1 0.2 1) ...) '(1 2 2 2.5 3.5 4) t nil)

CRSPPC - KI function to create a B-curve by splining

Syntax	(CRSPPC int address int address address) => (tag ifail)
Args	npts points nprops prop_array real_list_array
Returns	(B-curve ifail)
Syntax	(crsppc '(vector ...) '(token <real ... >) '(token ....) etc.) => tag
Args	points '(token <associated reals>) ...
Returns	B-curve
Example	(crsppc '((0 0 0) (0 0 1)) '(PAPRCU) '(PAPRKT 0.55 0.75))
Note	crsppc has an extendable argument list; as many tokens (together with any associated reals) as required may be supplied.

CRSPPS - KI function to create a B-surface by splining

Syntax	(CRSPPS int int address int address address) => (tag ifail)
Args	ncol nrow pts nprops prop_array real_list_array
Returns	(B-surface ifail)
Syntax	(crspps int int '(vector ...) '(token <real> ...) ...) => tag
Args	ncol nrow points token_&_reals ...
Returns	B-surface
Example	(crspps 2 2 '((0 0 0) (10 0 0) (0 10 0) (10 10 0)) '(PAPRCU) '(PAPRBL 0.4 0.5 0.6))))
Note	crspps takes an indefinite number of arguments. The user may supply any number of tokens with associated reals.

CRSPSO - KI function to create a spherical solid

Syntax	(CRSPSO vector double) => (tag ifail)
Args	centre radius
Returns	(body ifail)
Syntax	(crspso vector double) => tag

Args	centre radius
Returns	body

CRSPSU - KI function to create a spherical surface

Syntax	(CRSPSU vector double) => (tag ifail)
Args	centre radius
Returns	(surface ifail)
Syntax	(crspsu vector double) => tag
Args	centre radius
Returns	surface

CRSPTC - KI function to approximate a trimmed curve by SP-curves

Syntax	(CRSPTC tag tag double logical logical) => (int tag ifail)
Args	surface tr-curve tolerance degenerate sense
Returns	(n_curves sp_curves ifail)
Syntax	(crsptc tag tag double logical logical) => (tag ...)
Args	surface tr_curve tolerance degenerate sense
Returns	(sp_curve ...)

CRTOBY - KI function to create the topology of a body

Syntax	(CRTOBY int tag tag tag) => (tag int int ifail)
Args	body_type topol_types entity_ids children
Returns	(new_topol state_code fault_id ifail)
Syntax	(crtoby token '(token ...) '(int ...) '((<int> ...) ...)) => ((tag ...) token int)
Args	body_type '(topol_type ...) '(id ...) '((<child> ...))
Returns	((new_topol ...) state_code fault_id)
Example	(crtoby 'BYTYSO '(TYTOBY TYTOSH TYTOFA) '(1 2 3) '((2) (3) nil))

CRTOSO - KI function to create a toroidal solid

Syntax	(CRTOSO vector vector double double) => (tag ifail)
Args	centre axis majrad minrad
Returns	(body ifail)
Syntax	(crtoso vector vector double double) => tag

Args	centre axis majrad minrad
Returns	body

CRTOSU - KI function to create a toroidal surface

Syntax	(CRTOSU vector vector double double) => (tag ifail)
Args	centre axis majrad minrad
Returns	(surface ifail)
Syntax	(crtosu vector vector double double) => tag
Args	centre axis majrad minrad
Returns	surface

CRTRCU - KI function to create a trimmed curve

Syntax	(CRTRCU tag double double) => (tag ifail)
Args	curve start_parm end_parm
Returns	(trimmed_curve ifail)
Syntax	(crtrcu tag double double) => tag
Args	curve start_parm end_parm
Returns	trimmed_curve

CRTSFA - KI function to create a sheet body from a surface and trimmed SP-curve data

Syntax	(CRTSFA tag logical tag double double int address) => (tag tag int ifail)
Args	surface sense curves edge_tol face_tol n_opts options
Returns	(body face state ifail)
Syntax	(crtsfa tag logical '((tag ...) (tag ...)) double double (token ...)) => (tag tag token)
Args	surface sense list_of_lists_of_trimmed_curves edge_tol face_tol '(option ...)
Returns	body face state

DEFCON - KI function to make a connection between two entities

Syntax	(DEFCON tag tag) => (ifail)
Args	parent child
Returns	(ifail)
Syntax	(defcon tag tag) => t

Args	parent child
Returns	t

DEHOSH - KI function to delete a list of holes from a sheet

Syntax	(DEHOSH tag tag) => (ifail)
Args	sheet list_of_loops
Returns	(ifail)
Syntax	(dehosh tag '(tag tag)) => t
Args	sheet '(loop ...)
Returns	t
Example	(dehosh 18 29) (dehosh 18 '(29 85 121))

DELCON - KI function to break a connection between two entities

Syntax	(DELCON tag tag) => (ifail)
Args	parent child
Returns	(ifail)
Syntax	(delcon tag tag) => t
Args	parent child
Returns	t

DELENT - KI function to delete an entity

Syntax	(DELENT tag) => (ifail)
Args	entity
Returns	(ifail)
Syntax	(delent tag) => t
Args	entity
Returns	t
Note	delent does not raise an error if the tag is 0, neither does it attempt any action.

DELFAS - KI function to delete faces from a body

Syntax	(DELFAS tag int address) => (tag int tag ifail)
Args	face_list nactions actions

Returns	(body_list nbodies state_list ifail)
Syntax	(delfas '(tag ...) <'(token) . SLLOCP>) => ((tag token) ...)
Args	'(face ...) <'(action)>
Returns	((body state) ...)
Example	(delfas 19) (delfas 19 'SLLOGR) (delfas 19 '(SLLOGR SLLOLI))

DELIST - KI function to delete a list

Syntax	(DELIST tag) => (ifail)
Args	list
Returns	(ifail)
Syntax	(delist tag) => t
Args	list
Returns	t

DELIVL - KI function to delete items from a list

Syntax	(DELIVL tag int int) => (ifail)
Args	list start nitems
Returns	(ifail)
Syntax	(delivl list int int) => t
Args	list start nitems
Returns	t

DELSEN - KI function to delete a single geometric entity

Syntax	(DELSEN tag) => (ifail)
Args	geometry
Returns	(ifail)
Syntax	(delsen tag) => t
Args	geometry
Returns	t
Example	(delsen 26)

DETGEO - KI function to detach geometry from topology

Syntax	(DETGEO tag) => (ifail)
Args	topol
Returns	(ifail)
Syntax	(detgeo tag) => t
Args	topol
Returns	t

DLENFE - KI function to delete an entity from a feature

Syntax	(DLENFE tag tag) => (ifail)
Args	feature entity
Returns	(ifail)
Syntax	(dlenfe tag tag) => t
Args	feature entity
Returns	t

DLORPH - KI function to delete orphans

Syntax	(DLORPH int) => (ifail)
Args	type
Returns	(ifail)
Syntax	(dlorph <token>) => t
Args	<type>
Returns	t
Example	(dlorph 'TYENGE) (dlorph 'TYADLI) (dlorph)
Note	If the type token is not given, dlorph will delete orphans of both types.

ENBXEN - KI function to enquire box containing specified entity

Syntax	(ENBXEN tag) => (address ifail)
Args	entity
Returns	(box_array ifail)

Syntax	(enbxen tag) => (vector vector)
Args	entity
Returns	(lower_box_corner upper_box_corner)

ENBYTY - KI function to enquire body type

Syntax	(ENBYTY tag) => (int ifail)
Args	body
Returns	(type ifail)
Syntax	(enbyty tag) => token
Args	body
Returns	type

ENCONT - KI function to enquire containment of point

Syntax:	(ENCONT tag tag) => (int ifail)
Args:	point entity
Returns:	(enclosure ifail)
Syntax:	(encont vector tag) => token
Args:	point entity
Returns:	enclosure
Example:	(encont 7 8) (encont '(0 0 0) 7)
Note:	In fact encont accepts either a tag or a vector for its first argument (calling CRCAPO if required).

ENCUPA - KI routine to enquire curve parametrisation

Syntax:	(ENCUPA tag) => (address address tag int ifail)
Args:	curve
Returns:	(range_array bounds_array props nprops ifail)
Syntax:	(encupa tag) => ((real token) (real token) (token ...))
Args:	curve
Returns:	((bound type) (bound type) (property ...))
Example:	(encupa (t0 tag)) => ((-10000.0 PAPRIF) (10000.0 PAPRIF) (PAPRCN PAPRLI))

ENDFAT - KI routine to enquire the attribute type definition of an attribute

Syntax:	(ENDFAT tag) => (tag ifail)
Args:	attribute
Returns:	(att_type ifail)
Syntax:	(endfat tag) => tag
Args:	attribute
Returns:	att_type

ENDFNM - KI routine to enquire an attribute type definition from its name

Syntax:	(ENDFNM int string) => (tag ifail)
Args:	namlen name
Returns:	(att_type ifail)
Syntax:	(endfnm string) => tag
Args:	name
Returns:	att_type

ENDIPE - KI function to enquire discontinuities on a B-curve or B-surface

Syntax:	(ENDIPE tag int) => (int tag tag ifail)
Args:	geometry type_disc
Returns:	(ndisc u_or_v params ifail)
Syntax:	(endipe tag token) => (int (<token> ...) (<double> ...))
Args:	geometry type_disc
Returns:	((<u_or_v> ...) (<param> ...))
Example:	(endipe 20 'PADIG1)

ENEDTY - KI function to enquire edge type

Syntax:	(ENEDTY tag) => (int ifail)
Args:	edge
Returns:	(type ifail)
Syntax:	(enedty tag) => token
Args:	edge
Returns:	type

ENENTY - KI function to enquire entity type

Syntax:	(ENENTY tag int) => (address int ifail)
Args:	entity length
Returns:	(type_array ntypes ifail)
Syntax:	(enenty tag) => (token ...)
Args:	entity
Returns:	(type ...)

ENEQGE - KI function to enquire whether two geometries are equivalent

Syntax:	(ENEQGE tag tag) => (logical ifail)
Args:	geom1 geom2
Returns:	(equivalent ifail)
Syntax:	(eneqge tag tag) => logical
Args:	geom1 geom2
Returns:	equivalent

ENEXEN - KI function to enquire extreme point of entity

Syntax:	(ENEXEN tag vector vector vector) => (vector tag ifail)
Args:	entity dir1 dir2 dir3
Returns:	(position entity ifail)
Syntax:	(enexen tag vector <vector . (1 0 0)> <vector . (0 1 0)>) => (vector tag)
Args:	entity dir1 <dir2> <dir3>
Returns:	(position entity)
Note:	Defaults are provided as a convenience for the last two directions as these values are usually not of interest.

ENFAPR - KI function to enquire if a face is a parametrically rectangular

Syntax:	(ENFAPR tag) => (int address address ifail)
Args:	face
Returns:	(rectangular u_bound_arr v_bound_arr ifail)
Syntax:	(enfapr tag) => (logical (double double) (double double))

Args:	face
Returns:	(rectangular (u-low u-high) (v-low v-high))

ENLOTY - KI function to enquire loop type

Syntax:	(ENLOTY tag) => (int ifail)
Args:	loop
Returns:	(type ifail)
Syntax:	(enloty tag) => token
Args:	loop
Returns:	type

ENPAPC - KI function to find parameter of point on curve

Syntax:	(ENPAPC tag vector) => (double ifail)
Args:	curve position
Returns:	(parameter ifail)
Syntax:	(enpapc tag vector) => double
Args:	curve position
Returns:	parameter

ENPAPS - KI function to find parameters of point on surface

Syntax:	(ENPAPS tag vector) => (double double ifail)
Args:	surface position
Returns:	(u v ifail)
Syntax:	(enpaps tag vector) => (double double)
Args:	surface position
Returns:	(u v)

ENPBEN - KI function to calculate the parametric box of a given entity

Syntax:	(ENPBEN tag) => (address address ifail)
Args:	entity
Returns:	(ulimits vlimits ifail)
Syntax:	(enpben tag) => ((double double) (double double))

Args:	entity
Returns:	(lower_box_corner upper_box_corner)

ENPIFA - KI function to enquire if points in face

Syntax:	(ENPIFA tag int address int address int address address) => (address address ifail)
Args:	face n_parms parms n_pvecs pvecs n_opts opt_array opt_data
Returns:	(enclosure topology ifail)
Syntax:	(enpifa tag '(<double ... >) '(,vector ... >) '(<(token tag ...)>)) => ((token tag) ...)
Args:	face '(<parm ... >) '(<pvec ... >) '(<(option loop ...)>)
Returns:	((enclosure topology) ...)
Example:	(enpifa 79 '((-3 3 -12 0 -4 5 5 5) nil '((PFOPLO 78 494))) (enpifa 79 ' ((-3 3) (-5 5) nil nil) (enpifa 79 nil '(0 0 0) nil nil) (enpifa 79 nil '((0 0 0)) nil nil)
Note:	The lists of parameters and pvecs are flattened so any internal bracketing may be used.

ENPOGC - KI function to enquire point on general curve

Syntax:	(ENPOGC tag vector) => (vector vector vector double ifail)
Args:	curve position
Returns:	(tangent normal binormal curvature ifail)
Syntax:	(enpogc tag vector) => (vector vector vector double)
Args:	curve position
Returns:	(tangent normal binormal curvature)

ENPOGS - KI function to enquire point on general surface

Syntax:	(ENPOGS tag vector) => (vector vector vector double double ifail)
Args:	surface position
Returns:	(normal pdir1 pdir2 pcurvature1 pcurvature2 ifail)
Syntax:	(enpogs tag vector) => (vector vector vector double double)
Args:	surface position
Returns:	(normal pdir1 pdir2 pcurvature1 pcurvature2)

ENPOPC - KI function to evaluate point from curve parameter

Syntax:	(ENPOPC tag double int) => (address ifail)
Args:	curve parameter nderivs
Returns:	(derivative_array ifail)
Syntax:	(enpopc tag double <int>) => (vector ...)
Args:	curve parameter <nderivs . 0>
Returns:	(point ...)
Note:	The number of derivatives to enpopc is optional, and defaults to 0.

ENPOPS - KI function to evaluate a point from surface parameters

Syntax:	(ENPOPS tag double double int int logical) => (vector ifail)
Args:	surface u v nu_derivs nv_derivs nreq
Returns:	(<normal> ifail)
Syntax:	(enpops tag double double <int> <int>) => (vector ...)
Args:	surface u v <nu_derivs . 0> <nv_derivs . 0> <nreq . t>
Returns:	((point ...) <normal>)
Note:	The numbers of u and v derivatives in enpops default to 0. The default request for a normal is t.
	The function enpops manages workspace automatically, but ENPOPS requires workspace allocated of sufficient length to store the derivative vectors returned.

ENSHTY - KI function to enquire shell type

Syntax:	(ENSHTY tag) => (int ifail)
Args:	shell
Returns:	(type ifail)
Syntax:	(enshty tag) => token
Args:	shell
Returns:	type

ENSUPA - KI routine to enquire surface parametrisation

Syntax:	(ENSUPA tag) => (address address address address tag int tag int ifail)
Args:	surface

Returns:	(urange vrange ubound vbound uprops nuprops vprops nvprops ifail)
Syntax:	(ensupa tag) => (((real token) (real token) (token ...)) ((real token) (real token) (token ...)))
Args:	surface
Returns:	(((ubound type) (ubound type) (property ...)) ((vbound type) (vbound type) (property ...)))
Example:	(ensupa (surface tag)) => (((-10000.0 PAPRIF) (-10000.0 PAPRIF) (PAPRCN PAPRLI)) ((-10000.0 PAPRIF) (-10000.0 PAPRIF) (PAPRCN PAPRLI)))

ENVETY - KI function to enquire vertex type

Syntax:	(ENVETY tag) => (int ifail)
Args:	vertex
Returns:	(type ifail)
Syntax:	(envety tag) => token
Args:	vertex
Returns:	type

FIXIDS - KI function to fix identifiers in part

Syntax:	(FIXIDS tag) => (int tag tag tag ifail)
Args:	part
Returns:	(nfaults fixed_entities old_ids new_ids ifail)
Syntax:	(fixids tag) => ((<tag> ...) (<int> ...) (<int> ...))
Args:	part
Returns:	((<fixed_entity> ...) (<old_id> ...) (<new_id> ...))

FNENFE - KI function to find entity in a feature

Syntax:	(FNENFE tag tag) => (logical ifail)
Args:	feature entity
Returns:	(found ifail)
Syntax:	(fnenfe tag tag) => logical
Args:	feature entity
Returns:	found

GETMND - KI function to recover a faulty model

Syntax:	(GETMND int string int address) => (tag tag tag tag tag int int ifail)
Args:	key_length key nopts opt_array
Returns:	(part mends faults mended_components faulty_components ifail)
Syntax:	(getmnd string) => (tag (token ...) (token ...) (tag ...) (tag ...))
Args:	key
Returns:	(part (mend ...) (fault ...) (entity ...) (entity ...))

GETMOD - KI function to receive an archived model

Syntax:	(GETMOD int string) => (tag ifail)
Args:	length key
Returns:	(body ifail)
Syntax:	(getmod string) => tag
Args:	key
Returns:	body
Example:	(getmod 'gearwheel)
Notes:	Keys with embedded nulls are not supported from lisp.

GETSNP - KI function to restore a snapshot

Syntax:	(GETSNP int string int logical) => (ifail)
Args:	length filename histfl statfl
Returns:	(ifail)
Syntax:	(getsnp string <logical .t >) => t
Args:	filename <restore_interface_parms>
Returns:	t
Notes:	The logical flag controlling the restoration of interface parameters is optional and defaults to true.

GTINLI - KI function to get values from a list of integers

Syntax:	(GTINLI tag int int) => (int ifail)
Args:	list start nvals
Returns:	(vals ifail)
Syntax:	(gtinli tag int int) => (int ...)

Args:	list start nvals
Returns:	(value ...)
Notes:	Support function unlist provides convenient access to this function.
	The gtinli function manages workspace automatically, but GTINLI requires workspace supplied which is sufficiently long to store the integers output.

GTRLLI - KI function to get values from a list of reals

Syntax:	(GTRLLI tag int int) => (double ifail)
Args:	list start nvals
Returns:	(rvals ifail)
Syntax:	(gtrlli tag int int) => (double ...)
Args:	list start nvals
Returns:	(value ...)
Notes:	Support function unlist provides convenient access to this function.
	The gtrlli function manages workspace automatically. The GTRLLI function requires sufficient workspace to store the reals output.

GTTGLI - KI function to get values from a list of tags

Syntax:	(GTTGLI tag int int address) => (ifail)
Args:	list start nvals tag_array
Returns:	(tags ifail)
Syntax:	(gttgli tag int int) => (tag ...)
Args:	list start nvals
Returns:	(value ...)
Notes:	Support function unlist provides convenient access to this function.
	The gttgli function manages workspace automatically, but GTTGLI requires workspace allocated of sufficient length to store the tags output.

HOLLBY - KI function to hollow a solid body

Syntax:	(HOLLBY tag double logical tag tag tag double int) => (tag tag tag int ifail)
Args:	body offset check pierced_faces thickened_faces offsets tolerance max_faults
Returns:	(old_faces new_faces problem_tags state ifail)

Syntax:	(hollby tag double <logical . t> <(tag ...) . NULTAG> <(tag ...) . NUL TAG> <(double ...). NULTAG> <double . 1E-6><int .07)) => ((tag ...) (tag ...) (tag ...) token)
Args:	body offset <check> <pierced_faces> <thickened_faces> <offsets> <tolerance> <max_faults>
Returns:	((old_face ...) (new_face ...) (problem_tag ...) state)
Example:	(hollby 19 .1) (hollby 10 .1 t '(79) '(74 69) '(.2 .3))

IDATEN - KI function to enquire the attribute of a given type attached to an entity

Syntax:	(IDATEN tag tag) => (tag ifail)
Args:	entity att_type
Returns:	(attribute ifail)
Syntax:	(idaten tag tag) => tag
Args:	entity att_type
Returns:	attribute

IDATLS - KI function to enquire the attributes of a given type attached to an entity

Syntax:	(IDATLS tag tag) => (tag ifail)
Args:	entity att_type
Returns:	(attribute_list ifail)
Syntax:	(idatls tag tag) => (<tag> ...)
Args:	entity att_type
Returns:	(<attribute> ...)

IDATPA - KI function to identify all attributes of a given type in a part

Syntax:	(IDATPA tag tag) => (tag ifail)
Args:	part att_type
Returns:	(att_list ifail)
Syntax:	(idatpa tag tag) => (tag ...)
Args:	part att_type
Returns:	(attribute ...)

IDCCEN - KI function to identify common connected entities

Syntax:	(IDCCEN tag tag int) => (tag int ifail)
Args:	entity1 entity2 connection_type
Returns:	(entities n_entity ifail)
Syntax:	(idccen tag tag token) => (tag ...)
Args:	entity1 entity2 connection_type
Returns:	(common_entity ...)

IDCOEN - KI function to identify connected entities

Syntax:	(IDCOEN tag int) => (tag int ifail)
Args:	entity type
Returns:	(list nitems ifail)
Syntax:	(idcoen tag token) => (tag ...)
Args:	entity type
Returns:	(entity ...)
Example:	(idcoen 7 'TYTOFA)

IDCOFE - KI function to identify curve of edge

Syntax:	(IDCOFE tag) => (tag ifail)
Args:	edge
Returns:	(curve ifail)
Syntax:	(idcofe tag) => tag
Args:	edge
Returns:	curve

IDENID - KI function to identify entity by identifier

Syntax:	(IDENID tag int int) => (tag ifail)
Args:	part identifier type
Returns:	(entity ifail)
Syntax:	(idenid tag int token) => tag
Args:	part identifier type
Returns:	entity
Example:	(idenid 7 33 'TYTOFA)

IDFSEN - KI function to identify facesets of one or two bodies

Syntax:	(IDFSEN tag tag tag tag tag tag int address tag) => (tag tag tag tag tag tag ifail)
Args:	target tool targed tooled targvx toolvx nopts opt_array topol
Returns:	(targsu toolsu targbo toolbo targrj toolrj ifail)
Syntax:	(idfsen tag tag '(tag ...) '(tag ...) '(tag ...) '(tag ...) '((token < tag ... > => (((tag ...) ...) ((tag ...) ...) (tag ...) (tag ...) ((tag ...) ...) ((tag ...) ...)))...))
Args:	target tool '(edge ...) '(edge ...) '(vertex ...) '(vertex ...) '(option ...) < '(topology ...) >
Returns:	(((face ...) ...) ((face ...) ...) (edge ...) (edge ...) ((face ...) ...) ((face ...) ...))
Example:	(idfsen (b0 tag) (b1 tag) (e0 tag) nil nil (list 'IDOPSU (list 'IDOPFS (f1 tag))))
Notes:	A faceset is a collection of connected faces.

IDKYPA - KI function to identify keyed parts

Syntax:	(IDKYPA int string) => (tag tag int ifail)
Args:	length key
Returns:	(part_list state_list nparts ifail)
Syntax:	(idkypa string) => ((tag token) ...)
Args:	key
Returns:	((part state) ...)
Notes	Keys with embedded nulls are not supported via lisp.

IDLSID - KI function to identify entities by identifier

Syntax:	(IDLSID tag tag int) => (tag int ifail)
Args:	part identifier_list type
Returns:	(entities n_entity ifail)
Syntax:	(idlsid tag '(int ...) token) => (tag ...)
Args:	part '(identifier ...) type
Returns:	(entity ...)
Example:	(idlsid 7 '(33 35 37 65) 'TYTOFA)

IDNCEN - KI function to identify number of connected entities

Syntax:	(IDNCEN tag int) => (int ifail)
Args:	entity type
Returns:	(ifail)
Syntax:	(idncen tag token) => int
Args:	entity type
Returns:	n_connected

IDPOFV - KI function to identify point of vertex

Syntax:	(IDPOFV tag) => (tag ifail)
Args:	vertex
Returns:	(point ifail)
Syntax:	(idpofv tag) => tag
Args:	vertex
Returns:	point

IDSCEN - KI function to identify single connected entity

Syntax:	(IDSCEN tag int) => (tag ifail)
Args:	entity type
Returns:	(connected_entity ifail)
Syntax:	(idscen tag token) => tag
Args:	entity type
Returns:	connected_entity

IDSCLS - KI function to identify single connected entities of a list of entities

Syntax:	(IDSCLS tag int) => (tag int ifail)
Args:	entities type
Returns:	(connected_entities n_entity ifail)
Syntax:	(idscls '(tag ...) token)
Args:	'(entity ...) type
Returns:	(entity ...)

IDSOFF - KI function to identify surface of face

Syntax:	(IDSOFF tag) => (tag logical ifail)
Args:	face
Returns:	(surface reversed ifail)
Syntax:	(idsoff tag) => (tag logical)
Args:	face
Returns:	(surface reversed)

IMPRNT - KI function to imprint bodies or lists of faces

Syntax:	(IMPRNT tag tag int address) => (tag tag int tag tag int ifail)
Args:	target tool nopts opt_array
Returns:	(targed tooled nedges targvx toolvx nverts ifail)
Syntax:	(imprnt tag tag < '(token ...) >) => ((tag ...) (tag ...) (tag ...) (tag ...))
Args:	target tool < '(option ...) >
Returns:	((edge ...) (edge ...) (vertex ...) (vertex ...))
Example:	(imprint 17 77)

INCUCU - KI function to intersect two curves

Syntax:	(INCUCU tag address tag address tag address) => (tag tag tag tag int ifail)
Args:	curve bounds curve bounds surface box_array
Returns:	(intpts iparms1 iparms2 incods nintpt ifail)
Syntax:	(incucu tag vector vector tag vector vector <surface> <((double...)...)>) => ((token vector cvec cvec) ...)
Args:	curve start end curve start end <surface> <box.((-500 -500 -500) (500 500 500))>
Returns:	((intcod point iparm1 iparm2) ...)
Example:	(incucu 7 '(0 0 0) '(5 0 0) 34 '(4 0 0) '(1 2 3))
Notes:	The KI box is an array of doubles with all low coords first. The lisp box can be given as two extreme vectors or as 3 intervals. The box argument is optional in incucu. The default is for the whole of the standard size box.

INCUFA - KI function to intersect curve and face

Syntax:	(INCUFA tag address tag) => (tag tag tag tag tag int ifail)
Args:	curve bounds face

Returns:	(intpts cuparm suparm intcodes topol nintpt ifail)
Syntax:	(incufa tag vector vector tag) => ((token vector (double) (double double) tag) ...)
Args:	curve start end face
Returns:	((token point (cuparm) (suparm suparm) topol) ...)
Example:	(incufa 80 '(4 0 0) '(4 0 0) 23)

INCUSU - KI function to intersect a curve and a surface

Syntax:	(INCUSU tag address tag address) => (tag tag tag tag int ifail)
Args:	curve bounds surface box_array
Returns:	(intpts cuparm suparm incods nintpt ifail)
Syntax:	(incusu tag vector vector tag <((double ...)...)>) => ((token vector (double) (double double)) ...)
Args:	curve start end surface <box.((-500 -500 -500) (500 500 500))>
Returns:	((token point (cuparm) (suparm suparm)) ...)
Example:	(incusu 7 '(0 0 0) '(5 0 0) 34)
Notes:	The KI box is an array of doubles with all low coords first. The lisp box can be given as two extreme vectors or as 3 intervals. The box argument is optional in incusu. The default is for the whole of the standard size box.

INFAFA - KI function to intersect two faces

Syntax:	(INFAFA tag tag int address address) => (int tag int tag tag ifail))
Args:	face1 face2 n_opts options opt_data
Returns:	(n_points points n_curves curves types ifail)
Syntax:	(infafa tag tag '((token <real> ...) ...) => ((vector ...) ((tag token) ...)))
Args:	face1 face2 '((option <real> ...) ...)
Returns:	((point ...) ((curve type) ...))

INSUFA - KI function to intersect a surface with a face

Syntax:	(INSUFA tag tag int address address) => (int tag int tag tag ifail))
Args:	surface face n_opts options opt_data
Returns:	(n_points points n_curves curves types ifail)
Syntax:	(insufa tag tag '((token <real> ...) ...) => ((vector ...) ((tag token) ...)))
Args:	surface face '((option <real> ...) ...)
Returns:	((point ...) ((curve type) ...))

INSUSU - KI function to intersect two surfaces

Syntax:	(INSUSU tag tag int address address) => (int tag int tag tag ifail)
Args:	surf1 surf2 n_opts options real_data
Returns:	(n_points points n_curves curves types ifail)
Syntax:	(insusu tag tag '((token <real> ...) ...) => ((vector ...) ((tag type) ...)
Args:	surf1 surf2 '((options <points> ...) ...)
Returns:	((point ...) ((curve int_type) ...)

INTBYS - KI function to intersect bodies

Syntax:	(INTBYS tag tag) => (tag int ifail)
Args:	body tool_list
Returns:	(assembly nbodies ifail)
Syntax:	(intbys tag '(tag ...)) => (tag int)
Args:	target '(tool ...)
Returns:	(assembly nbodies)

KABORT - KI function to abort an interrupted Kernel operation

Syntax:	(KABORT int) => (ifail)
Args:	reason
Returns:	(ifail)
Syntax:	(kabort <token>) => t
Args:	<reason>
Returns:	t
Example:	(kabort 'SLABRE)

KNITEN - KI function to “knit together” bodies by fusing coincident edges

Syntax:	(KNITEN int tag tag tag logical) => (int tag int ifail)
Args:	body_type target edge_list1 edge_list2 sort_shells
Returns:	(state failed_edges nfailed ifail)
Syntax:	(kniten token tag '(tag ...) '(tag ...)<logical.nil>) => (token (<tag> ...))
Args:	body_type target (edge ...) (edge ...)<sort_shells>
Returns:	(state (<edge> ...))

LEVASS - KI function to level assemblies

Syntax:	(LEVASS tag) => (tag ifail)
Args:	assembly
Returns:	(new_assembly ifail)
Syntax:	(levass tag) => tag
Args:	assembly
Returns:	new_assembly

MASSPR - KI function to compute mass and related property calculations

Syntax:	(MASSPR tag int address double) => (tag tag tag tag tag ifail)
Args:	entity_list nopts opt_array accuracy
Returns:	(real_list real_list real_list real_list real_list ifail)
Syntax:	(masspr '(tag ...) '(token ...) double) => ((real...) (real...) (real ...) (real ...) (real...))
Args:	'(entity ...) '(options ...) accuracy
Returns:	((periph ...) (amount ...) (mass...) (cog ...) (inertia ..))
Example:	(masspr 7 '(MAOPCG MAOPEM) 0.95)

MENDEN - KI function to mend a model

Syntax:	(MENDEN tag logical) => (tag tag tag tag tag tag tag int int int int int int ifail)
Args:	body replace_all
Returns:	(fixed_edges fixed_vertices faulty_edges faulty_vertices edge_faults vertex_faults old_geom nfixed nfixvx nftyed nftyvx nold ifail)
Syntax:	(menden tag <logical . nil>) => ((<tag> ...) (<tag> ...) (<tag> ...) (<tag> ...) (<token> ...) (<token> ...) (<tag> ...) token)
Args:	body <replace_all>
Returns:	((<fixed_edge> ...) (<fixed_vertex> ...) (<faulty_edge> ...) (<faulty_vertex> ...) (<edge_fault> ...) (<vertex_fault> ...) (<old_geom> ...) body_state)
Example:	(menden 17)

MERGEN - KI function to remove redundant topology from an entity

Syntax:	(MERGEN tag) => (ifail)
Args:	entity
Returns:	(ifail)

Syntax:	(mergen tag) => token
Args:	entity
Returns:	ifail_token
Notes:	Unlike most lower-case functions, mergen returns the ifail explicitly. This is most likely to be KI_no_errors or KI_non_mergeable

NEGENT - KI function to negate (reverse) an entity

Syntax:	(NEGENT tag) => (ifail)
Args:	entity
Returns:	(ifail)
Syntax:	(negent tag) => tag
Args:	entity
Returns:	entity

OFFABY - KI function to offset the faces of a solid or sheet body

Syntax:	(OFFABY tag double logical tag tag tag double int) => (tag int ifail)
Args:	body offset check pierced_faces thickened_faces offsets tolerance max_faults
Returns:	(problem_tags state ifail)
Syntax:	(offaby tag double <tokens . t> <(tag ...) . NULTAG> <(tag ...) . NUL TAG> <(double ...) . NULTAG>) <double . 1E-6><int .07>=> ((tag ...) token)
Args:	body offset <check> <pierced_faces> <thickened_faces> <offsets> <tolerance> <max_faults>
Returns:	((problem_tag ...) state)
Example:	(offaby 19 .1) (offaby 19 .1 t '(79) '(74 69) '(.2 .3))

OUATDF - KI function to output an attribute type definition

Syntax:	(OUATDF tag int) => (address int tag tag ifail)
Args:	att_type bufsiz
Returns:	(name namlen opt_list data_list ifail)
Syntax:	(ouatdf tag) => (string (token ...) ...)
Args:	att_type
Returns:	(name (option data ...) ...)

Example:	(ouatdf 133) => (Colour (ATOPCL RQAC01) (ATOPOW TYTOFA) (ATOPFL RQAPCS))
	ouatdf is restricted to a buffer size of eighty characters.

OUBBCO - KI function to output bulletin board controls

Syntax:	(OUBBCO) => (tag tag int tag int ifail)
Args:	- none -
Returns:	(entity_types events nentities options nopts ifail)
Syntax:	(oubbco) = ((< (token (token ...)) ... >) (token < ... >))
Args:	- none -
Returns:	((< (entity_type (event ...)) ... >) (option < ... >))
Example:	(oubbco) => (BBOPOF) (oubbco) => (((TYTOFA (BBEVCR BBEVDE BBEVCH)) (TYADAD (BBEVCR))) (BBOPON))

OUBBEV - KI function to output full bulletin board information

Syntax:	(OUBBEV logical) => (int tag tag int tag tag tag ifail)
Args:	empty_board
Returns:	(nevents events nperev nents ents enttyp usflds ifail)
Syntax:	(oubbev logical) => ((token (tag token <(int ...)>) ...) ...)
Args:	empty_board
Returns:	((event (entity type <usflds>)) ...)

OUBLSS - KI function to output blend surface definition

Syntax:	(OUBLSS tag) => (int tag tag tag tag tag tag int tag tag tag int logical ifail)
Args:	surface
Returns:	(type sf1 sf2 sf3 iparm12 iparm23 iparm31 nipars rpars12 rpars23 rpars31 nrpars sense ifail)
Syntax:	(oublss tag) => (token tag tag tag (int...) (int...) (int...) (double...) (double...) (double...) logical)
Args:	surface
Returns:	(type sf1 sf2 sf2 (iparm12...) (iparm23...)(iparm31...)(rparm12...) (rparm23...)(rparm31...) sense)

OUBSCU - KI function to output a curve in B-spline form

Syntax:	(OUBSCU tag double int address) => (tag int int int tag tag int ifail)
Args:	curve tolerance option_count option_arr
Returns:	(ctrls dim order n_cntrl knot_vector props n_props ifail)
Syntax:	(oubscu tag double (token ...)) => ((double ...) ...) int int int (double ...) (token ...))
Args:	curve tolerance '(<option> ...)
Returns:	control_points dimension order knot_vector (property ...)

OUBSED - KI function to output the curve of an edge in B-spline form

Syntax:	(OUBSED tag double int address) => (tag int int int tag tag int ifail)
Args:	edge tolerance nopt opt_array
Returns:	(points dim order ncontrol knots properties nprops)
Syntax:	(oubsed tag double '(token ...)) => (((double ...) ...) int (double ...) (token ...))
Args:	edge tolerance '(option ...)
Returns:	((point ...) dim order (knot ...) (property ...))
Notes:	The list of data points returned may be of dimension 3 or 4 and are packed in lists of rows.

OUBSFA - KI function to output the surface of a face in B-spline form

Syntax:	(OUBSFA tag double int address) => (tag int int int int int tag tag tag int ifail)
Args:	face tol nopts option_array
Returns:	(points dim uorder vorder ncol nrow uknots vknots props nprops ifail)
Syntax:	(oubsfa tag double '(token ...)) => ((((double ...)...)...) int int int int int (double ...) (double...) (token ...))
Args:	face tolerance '(option ...)
Returns:	(((point ...)...) dim uorder vorder ncol nrow (uknot ...) (vknot ...) (property ...))
Example:	(oubsfa 19 0.5 '(SROPCU SROPNR))
Notes:	The list of data points returned may be of dimension 3 or 4 and are packed in lists of rows.

OUBSPC - KI function to output a B-curve in B-spline form

Syntax:	(OUBSPC tag) => (tag int int int tag tag int ifail)
Args:	B-curve
Returns:	(controls dim order ncontrol knots properties nprops ifail)
Syntax:	(oubspc tag) => ((double ...)...) int int (double ...) (token ...))
Args:	B-curve
Returns:	((point ....)...) dim order (knot ...) (property ...))

OUBSPS - KI function to output a B-surface in B-spline form

Syntax:	(OUBSPS tag) => (tag int int int int int tag tag tag int logical ifail)
Args:	B-surface
Returns:	(controls dim uord vord ncol nrow uknots vknots properties nprops sense ifail)
Syntax:	(oubsps tag) => (((double ...)...) int int int int int (double...) (double ...) (token ...) logical)
Args:	B-surface
Returns:	((points ...) dim uord vord ncol nrow (uknot...) (vknot...) (property ...) sense)

OUBSSU - KI function to output a region of a surface in B-spline form

Syntax:	(OUBSSU tag address address double int address) => (tag int int int int int tag tag tag int ifail)
Args:	surface urange_arr vrange_arr tolerance n_options option_arr
Returns:	(controls dimension uorder vorder ncol nrow uknots vknots nprops proper ties ifail)
Syntax:	(oubssu tag (double double) (double double) double (token ...)) => ((double ...) int int int int int (double ...) (double ...) (token ...))
Args:	surface u_range v_range tolerance '(option ...)
Returns:	(controls dimension uorder vorder ncol nrow uknots vknots properties)

OUCOCU - KI function to output coordinates on curve

Syntax:	(OUCOCU tag vector vector double double double) => (tag int ifail)
Args:	curve start end ctol atol stol
Returns:	(vec_list npos ifail)
Syntax:	(oucocu tag vector vector double double double) => (vector ...)

Args:	curve start end ctol atol stol
Returns:	(coord ...)

OUCPCU - KI function to output a constant parameter line curve

Syntax:	(OUCPCU tag) => (tag int double int ifail)
Args:	curve
Returns:	(surface u_or_v param sense ifail)
Syntax:	(oucpcu tag) => (tag token double logical)
Args:	curve
Returns:	(surface u_or_v param sense)
Example:	(oucpcu 17)

OUEXSU - KI function to output an extruded surface

Syntax:	(OUEXSU tag) => (tag vector int ifail)
Args:	surface
Returns:	(curve path sense ifail)
Syntax:	(ouexsu tag) => (tag vector logical)
Args:	surface
Returns:	(curve path sense)

OUFEAT - KI function to output items in feature

Syntax:	(OUFEAT tag) => (int tag int ifail)
Args:	feature
Returns:	(type list nitems ifail)
Syntax:	(oufeat tag) => (token (tag ....))
Args:	feature
Returns:	(type (entity ...))

OUFGCU - KI function to output a foreign geometry curve

Syntax:	(OUFGCU tag int) => (address int tag tag int tag ifail)
Args:	curve nchars
Returns:	(key keylen ivals rvals sense transform ifail)
Syntax:	(oufgcu tag int) => (tag (<int> ...) (<double> ...) logical tag)

Args:	curve nchars
Returns:	(key (<ival> ...) (<rval> ...) sense transform)
Example:	(oufgcu 84 3)

OUFGSU - KI function to output a foreign geometry surface

Syntax:	(OUFGSU tag int) => (address int tag tag int tag ifail)
Args:	surface nchars
Returns:	(key keylen ivals rvals sense transform ifail)
Syntax:	(oufgsu tag int) => (tag (<int> ...) (<double> ...) logical tag)
Args:	surface nchars
Returns:	(key (<ival> ...) (<rval> ...) sense transform)
Example:	(oufgsu 84 3)

OUFINF - KI function to output information about the specified file

Syntax:	(OUFINF int string int int int) => (int double string ifail)
Args:	length name guise format selection
Returns:	(ival rval sval nstring ifail)
Syntax:	(oufinf string token token) => int
Args:	name guise format
Returns:	version_file_created
Example:	(oufinf “cube” 'FFCXMT 'FFTEXT)

OUGEEF - KI function to output geometry of edge or fin

Syntax:	(OUGEEF tag logical) => (tag int vector double vector double int ifail)
Args:	edge_or_fin parms
Returns:	(curve curve_type start start_t end end_t sense ifail)
Syntax:	(ougeef tag (logical . t)) => (tag token vector double vector double logical)
Args:	edge_or_fin parms
Returns:	(curve curve_type start start_t end end_t sense)
Example:	(ougeef 24) (ougeef 24 nil)

OUGESU - KI function to output generated surface

Syntax:	(OUGESU tag) => (tag int tag int tag tag tag tag int tag tag int logical ifail)
Args:	surface
Returns:	(types ntypes codes ncodes ints reals geoms singc nsingc singp unused nsingp sense ifail)
Syntax:	(ougesu tag) => ((token...) (token...) (int...) (double...) (tag...) (vector...) logical)
Args:	surface
Returns:	((type...) (code...) (int ...) (double...) (geometry...) (sing_curve...) (sing_point...) sense)

OUIDEN - KI function to output identifier of entity

Syntax:	(OUIDEN tag) => (int ifail)
Args:	entity
Returns:	(identifier ifail)
Syntax:	(ouiden tag) => int
Args:	entity
Returns:	identifier

OUIDLS - KI function to output identifiers of a list of entities

Syntax:	(OUIDLS tag) => (tag int ifail)
Args:	list
Returns:	(id_list nitems ifail)
Syntax:	(ouidls '(tag ...)) => (int ...)
Args:	'(entity ...)
Returns:	(int ...)

OUINTP - KI function to output interface parameters

Syntax:	(OUINTP int) => (int double ifail)
Args:	option
Returns:	(ival rval ifail)
Syntax:	(ouintp <token>) => (int string) or ((int string) ...)
Args:	<option>
Returns:	(ival description)

Example:	(ouintp 'SLIPLC) (ouintp)
Notes:	The return value is given as a list containing the integer value (as returned directly by the KI), and the string describing its meaning. Any programming applications should use the actual integer parameter; the string message is only meant as mnemonic aid.
	The parameter is optional, if it is omitted a list of all current option settings will be returned, each element of the list formed as described above.

OULERR - KI function to enquire on the most recent KI-error

Syntax:	(OULERR int) => (int string int ifail)
Args:	option
Returns:	(ival sval length ifail)
Syntax:	(oulerr <token>) => (string string (string) string) or (int string)
Args:	<option**>**
Returns:	(ifail_mnemonic KI_routine (bad_arg) error_message) or (ival sval)
Example:	(oulerr 'SLEREX) (oulerr)
Notes:	The parameter is optional. If it is omitted a list of strings is created to return a generic message containing information from several options combined. If it is supplied the ival and sval for that option are returned.

OUMODP - KI function to output modeling parameter

Syntax:	(OUMODP int) => (int double ifail)
Args:	option
Returns:	(ival rval ifail)
Syntax:	(oumodp token) => double
Args:	option
Returns:	resolution

OUOFSU - KI function to output an offset surface

Syntax:	(OUOFSU tag) => (tag double int ifail)
Args:	surface
Returns:	(underlying_surface distance sense ifail)
Syntax:	(ouofsu tag) => (tag double logical)

Args:	surface
Returns:	(underlying_surface distance sense)
Example:	(ouofsu 20)

OUPART - KI function to output key and state of a part

Syntax:	(OUPART tag int) => (int address int ifail)
Args:	part buffer_length
Returns:	(length char_array state ifail)
Syntax:	(oupart tag) => (string token)
Args:	part
Returns:	(key state)
Notes:	The key may be the empty string (“”) for anonymous parts.

OUPWPC - KI function to output B-curve in piecewise form

Syntax:	(OUPWPC tag token) => (tag int int int ifail)
Args:	paracurve basis
Returns:	(coeffs dim order nsegments ifail)
Syntax:	(oupwpc tag token) => ((double...)...) int int int)
Args:	paracurve basis
Returns:	((points ...) dim order nsegments)

OUPWPS - KI function to output B-surface in piecewise form

Syntax:	(OUPWPS tag token) => (tag int int int int int logical ifail)
Args:	parasurface basis
Returns:	(coeffs dim uord vord ncol nrow sense ifail)
Syntax:	(oupwps tag token) => (((double ...)...) int int int int int logical)
Args:	parasurface basis
Returns:	(((point...)...) dim uord vord ncol nrow sense)

OURVSU - KI function to output data on a surface of revolution

Syntax:	(OURVSU tag) => (tag vector vector int int tag ifail)
Args:	surface

Returns:	(curve point axis sense n_sings parameter_range ifail)
Syntax:	(ourvsu tag) => (tag vector vector logical int (double double))
Args:	surface
Returns:	(curve point axis sense n_sings (t_low t_high))

OUSPCU - KI function to output an SP-curve in B-spline form

Syntax:	(OUSPCU tag) => (tag tag int int int tag ifail)
Args:	SP-curve
Returns:	(surface ctrls dim order nctrl knots ifail)
Syntax:	(ouspcu tag) => (tag int int int ((double ...) ...) (double ...))
Args:	SP-curve
Returns:	(surface dim order nctrl ((ctrl_pt ...) ...) (knot ...))

OUSPPC - KI function to output a B-curve as spline points

Syntax:	(OUSPPC tag) => (tag int tag tag int ifail)
Args:	B-curve
Returns:	(points npts prop_list <real_list>_list nprops ifail)
Syntax:	(ousppc tag) => ((vector ...) ((token <real ...>) ...))
Args:	B-curve
Returns:	((point ...) ((code data ...) ...))

OUSPPS - KI function to output a B-surface as spline points

Syntax:	(OUSPPS tag) => (tag int int tag tag int int ifail)
Args:	B-surface
Returns:	(points ncol nrow props <data_list>_list nprops sense ifail)
Syntax:	(ouspps tag) => ((vector ...) ((token <double ...>) ...) int int logical)
Args:	B-surface
Returns:	((point...) ((code <data...>) ...) ncol nrow sense)

OUSTAT - KI function to output information about the current state of the kernel

Syntax:	(OUSTAT int) => (int double ifail)
Args:	option

Returns:	(ival rval ifail)
Syntax:	(oustat <token>) => (int string) or ((int string) ...)
Args:	<option>
Returns:	(ivalue string)
Example:	(oustat 'SLSTAR) (oustat)
Notes:	Each option is returned in two ways, firstly as the integer directly provided by the KI, and secondly as a string describing its meaning. Only the first should be used by programming applications as the second is only intended as a mnemonic guide.
	If the option is omitted, all the possible option requests are returned in a list with each element constructed as described above.

OUTATT - KI function to output an attribute

Syntax:	(OUTATT tag int) => (tag tag tag tag address ifail)
Args:	attribute bufsiz
Returns:	(owner int_list real_list int_list char_list ifail)
Syntax:	(outatt tag) => (tag (int ...) (double ...) (string ...))
Args:	attribute
Returns:	(entity (int_fields) (double_fields) (string_fields))
Notes:	The total length of the string_fields is restricted to 400 characters.

OUTBUB - KI function to output rudimentary bulletin board information

Syntax:	(OUTBUB logical) => (tag tag tag tag tag tag tag tag tag ifail)
Args:	empty
Returns:	(new changed deleted new_types changed_types deleted_types new_user changed_user deleted_user ifail)
Syntax:	(outbub logical) => ((tag...) (tag...) (tag ...) (token...) (token...) (token...) ((int...)...) ((int...)...) ((int...)))
Args:	empty
Returns:	((new...) (changed...) (deleted...) (new_type..) (changed_type...) (deleted_type...) (new_user...) (changed_user ...) (deleted_user...)

OUTCUR - KI function to output curve definition

Syntax:	(OUTCUR tag) => (int vector vector vector double double ifail)
Args:	curve
Returns:	(type vec1 vec2 vec3 real1 real2 ifail)
Syntax:	(outcur tag) => (token vector vector <vector> <double> <double>)
Args:	curve
Returns:	(type vec1 vec2 <vec3> <real1> <real2>)
Notes:	Only the appropriate elements are returned with the type token.

OUTLEN - KI function to output the tolerance value associated with an entity

Syntax:	(OUTLEN tag) => (double ifail)
Args:	entity
Returns:	(tolerance ifail)
Syntax:	(outlen tag) => double
Args:	entity
Returns:	tolerance
Example:	(outlen 20)

OUTPOI - KI function to output a point

Syntax:	(OUTPOI tag) => (int vector ifail)
Args:	point
Returns:	(type coords ifail)
Syntax:	(outpoi tag) => vector
Args:	point
Returns:	coords

OUTRAN - KI function to output transformation

Syntax:	(OUTRAN tag) => (address ifail)
Args:	transform
Returns:	(real_array ifail)
Syntax:	(outran tag) => ((double...) (double...) (double...) (double...))
Args:	transform
Returns:	((transformation_matrix_coeff ...) ...)

Notes:	outran returns 16 real coefficients in a list of 4 x 4.

OUTRCU - KI function to output a trimmed curve

Syntax:	(OUTRCU tag) => (tag vector vector double double ifail)
Args:	trimmed_curve
Returns:	(curve start_pos end_pos start_parm end_parm ifail)
Syntax:	(outrcu tag) => (tag vector vector double double)
Args:	trimmed_curve
Returns:	(curve start_pos end_pos start_parm end_parm)

OUTSFA - KI routine to output trimmed surfaces

Syntax:	(OUTSFA tag int address address) => (tag int tag tag tag ifail)
Args:	faces nopts opt_array data_array
Returns:	(surface no_of_trimming_sets list_list_<list>_trimmed_sp_curves list_list_<list>_geom list_list_<list>_topol ifail)
Syntax:	(outsfa tag '((token <data>) ...)) => (tag int (<(((tag <...>) <(tag <...>)> < (tag <...>)>) ...) ... >))
Args:	faces '((option <data>) ...)
Returns:	(surface no_of_trimming_sets (<trimming_set> ...))
Examples:	spherical face: (outsfa (f0 tag)) => (33 0 nil)
	circular face: (outsfa (f0 tag)) => (77 1 ((((99))))) (outsfa (f0 tag) '((SROPBS 0.1) (SROPCU) (SROPNG) (SROPNT))) => (47 1 ((((99) (101) (89)))))
	cylinder face (crossing parameter seam): (outsfa (f0 tag)) => (823 2 ((((842 845 848 851)) ((830 833 836 839)))))

Notes:	The geom and topol lists are optional and are _missing_ (i.e. not even nil if not requested). So the third element of the output is a list of trimming_sets: i.e. (trimming_set ...) where a trimming_set is a list of trimmed_loops, so we have ((trimmed_loop ...) ...) where each trimmed loop is as follows:
	options SROPNG and SROPNT: ■ trimmed_loop: ((sp_curve ...) (geom ...) (topol ...)) ■ overall: ((((sp_curve ...) (geom ...) (topol ...)) ...) ...)
	option SROPNG: ■ trimmed_loop: ((sp_curve ...) (geom ...)) ■ overall: ((((sp_curve ...) (geom ...)) ...) ...)
	option SROPNT: ■ trimmed_loop: ((sp_curve ...) (topol ...)) ■ overall: ((((sp_curve ...) (topol ...)) ...) ...)
	neither option SROPNG nor SROPNT: ■ trimmed_loop: ((sp_curve ...)) ■ overall: ((((sp_curve ...)) ...) ...)

OUTSUR - KI function to output surface

Syntax:	(OUTSUR tag) => (int vector vector double double logical ifail)
Args:	surface
Returns:	(type vec1 vec2 double1 double2 sense ifail)
Syntax:	(outsur tag) => (token <vector> <vector> <double> <double> logical)
Args:	surface
Returns:	(type <vec1> <vec2> <double1> <double2> sense)
Notes:	outsur only returns the relevant surface parameters. The type and sense are always returned.

OUUFEN - KI function to return the user field of an entity

Syntax:	(OUUFEN tag) => (array ifail)
Args:	entity
Returns:	(workspace_address ifail)
Syntax:	(ouufen tag) => (int ...)
Args:	entity
Returns:	int_list

PICKEN - KI function to pick entities inside a cylindrical volume

Syntax:	(PICKEN tag tag vector vector double int int) => (int tag tag tag tag ifail)
Args:	parts transforms point axis radius opt type
Returns:	(nhit items indices distances points ifail)
Syntax:	(picken '(tag ...) '(tag ...) vector vector double token token) => ((tag int double vector) ...)
Args:	'(part...) '(transform...) point axis radius type <opt . SLPKIR>
Returns:	((item owner_index distance point) ...)
Notes:	Either a list of parts or a single part tag may be passed to picken. If no transforms are required they may be supplied as nil.

PIERCE - KI function to remove face from a sheet

Syntax:	(PIERCE tag) => (ifail)
Args:	face
Returns:	(ifail)
Syntax:	(pierce tag) => t
Args:	face
Returns:	t

PTENFE - KI function to put entities into feature

Syntax:	(PTENFE tag tag) => (ifail)
Args:	feature entity
Returns:	(ifail)
Syntax:	(ptenfe tag '(tag...)) => t
Args:	feature '(entity ...)
Returns:	t
Notes:	ptenfe accepts either a single entity or a list of entities, whereas PTENFE will accept only a single entity per call.

PTINLI - KI function to put values into a list of integers

Syntax:	(PTINLI tag int int address) => (ifail)
Args:	list start nitems integer_array
Returns:	(ifail)

Syntax:	(ptinli tag int '(int ...)) => tag
Args:	list start '(value ...)
Returns:	list
Notes:	Support function “enlist” provides convenient access to this function. ptinli echoes the original list as a return value.

PTRLLI - KI function to put values into a list of reals

Syntax:	(PTRLLI tag int int address) => (ifail)
Args:	list start nitems double_array
Returns:	(ifail)
Syntax:	(ptrlli tag int '(double ...)) => tag
Args:	list start '(value ...)
Returns:	list
Notes:	Support function “enlist” provides convenient access to this function. ptrlli echoes the original list as a return value.

PTTGLI - KI function to put values into a list of tags

Syntax:	(PTTGLI tag int int address) => (ifail)
Args:	list start ntags tag_array
Returns:	(ifail)
Syntax:	(pttgli tag int '(tag ...)) => tag
Args:	list start '(tag...)
Returns:	list
Example:	(pttgli 546 1 (list (b0 tag) (c0 tag)))
Notes:	Support function “enlist” provides convenient access to this function. pttgli echoes the original list as a return value.

RAYFIR - KI function to intersect ray with bodies

Syntax:	(RAYFIR tag tag int vector vector) => (int tag tag tag ifail)
Args:	parts transforms nhits point direction
Returns:	(nhits points faces indices ifail)
Syntax:	(rayfir '(tag ...) '(tag ...) int vector vector) => (vector tag int) ...)

Args:	parts transforms nhits point direction
Returns:	((coord face owner_index) ...)

REDINS - KI function to redirect an instance

Syntax:	(REDINS tag tag) => (ifail)
Args:	instance part
Returns:	(ifail)
Syntax:	(redins tag tag) => t
Args:	instance part
Returns:	t

REEDSH - KI function to replace the edges of a sheet body

Syntax:	(REEDSH tag address address) => (ifail)
Args:	sheet u_range v_range
Returns:	(ifail)
Syntax:	(reedsh tag '(double double) '(double double)) => t
Args:	sheet '(u_low u_high) '(v_low v_high)
Returns:	t
Example:	(reedsh 21 '(.1 .2) '(3 4)) (reedsh 21 (list u_low v_low) '(3 4))

RESUSH - KI function to replace the surface of a sheet body

Syntax:	(RESUSH tag tag double) => (tag int ifail)
Args:	sheet surface tolerance
Returns:	(edges nedges ifail)
Syntax:	(resush tag '(tag tag))) => <(tag ...)>
Args:	sheet surface tolerance
Returns:	<(edge ...)>
Example:	(resush 19 29 .1)

RETLEN - KI function to restore Parasolid tolerance to the supplied edge

Syntax:	(RETLEN tag) => (int ifail)
Args:	edge
Returns:	(status ifail)

Syntax:	(retlen tag) => token
Args:	edge
Returns:	status
Example:	(retlen 76) => RTTLOK

RMFASO - KI function to remove faces into new solids

Syntax:	(RMFASO tag int int) => (tag int tag tag int tag ifail)
Args:	faces actpar actoff
Returns:	(parents nparents parent_states offspring noff offspring_states)
Syntax:	(rmfaso '(tag ...) token token) => ((tag token) ...) ((tag token) ...))
Args:	faces actpar actoff
Returns:	(((parent state) ...) ((offspring state) ...))

ROLBFN - KI function to roll back or forward by n steps between roll-marks

Syntax:	(ROLBFN int) => (int ifail)
Args:	nsteps
Returns:	(nsteps ifail)
Syntax:	(rolbfn int) => int
Args:	nsteps
Returns:	nsteps

ROLBLM - KI function to roll back changes since last roll-mark

Syntax:	(ROLBLM) => (ifail)
Args:	-none-
Returns:	(ifail)
Syntax:	(rolblm) => t
Args:	-none-
Returns:	t

ROLSMK - KI function to set a roll-back mark

Syntax:	(ROLSMK) => (ifail)
Args:	-none-

Returns:	(ifail)
Syntax:	(rolsmk) => t
Args:	-none-
Returns:	t

RRFCET - KI function to generate faceted rendering

Syntax:	(RRFCET int address address tag tag) => (ifail)
Args:	nopts opt_array opt_data entities transforms
Returns:	(ifail)
Syntax:	(rrfcet '((token <double>...) ...) '(tag ...) '(tag ...)) => t
Args:	'((option <data> ...) ...) '(entity ...) '(transform ...)
Returns:	t
Example:	(rrfcet '((RROPFS 5 0.6) (RROPCV)) 7 nil)
Notes:	Each rrfcet option may be accompanied by real data values. Either a single entity or a list of entities may be faceted. The transform list may be supplied as nil.

RRHIDL - KI function to generate hidden line data

Syntax:	(RRHIDL int address address tag tag address) => (ifail)
Args:	nopts opts opt_data entities transforms view_matrix
Returns:	(ifail)
Syntax:	(rrhidl '((token <double> ...) ...) '(tag...) '(tag...) '(double...)) => t
Args:	options entities transforms view_matrix
Returns:	t
Example:	(rrhidl nil 410 nil Vw) (rrhidl '((RROPIV) (RROPPH 1 (0 0 1)) '(7 410) nil Vw)
Notes:	In rrhidl each option is passed in list together with any associated data parameters. A single body or a list of bodies may be passed. The transform list may be left as null. 16 doubles must be supplied to identify the viewing transform.

RRPIXL - KI function to generate shaded picture (pixel) data

Syntax:	(RRPIXL int address address tag tag address tag tag) => (ifail)
Args:	nopts opts optdata ents transf view_matrix pixel lights
Returns:	(ifail)

Syntax:	(rrpixl '((token <double> ...)...) '(tag ...) '(tag ...) '(double ...) '(double ...) ((double ...)...)) => t
Args:	'((option <data> ...)...) '(entity ...) '(transform ...) '(view_matrix ...) '(npix_x npix_y pix_size_x pix_size_y org_x org_y) '((light_type (red green blue) (x y z)) ...))
Returns:	t
Example:	(rrpixl '((RROPDM) (RROPSF 0.5 0.5 0 0 1)) '(7 40) nil Vw '(50 50 1 1 0 0) '((3 (0.3 0.3 0.3) (0.0 0.0 0.0)) (1 (1 0 0) (0 0 1))))

RRVDEP - KI function to generate view dependent rendering data

Syntax:	(RRVDEP int address address tag tag address) => (ifail)
Args:	nopts opt_array opt_data entities transforms view
Returns:	(ifail)
Syntax:	(rrvdep '((token) ...) '(tag ...) '(tag ...) '(double ...)) => t
Args:	'((option) ...) '(entity ...) '(transform ...) view)
Returns:	t
Example:	(rrvdep '((RROPSI)) 7 nil Vw)
Notes:	Some rrvdep options are accompanied by real data. Either a single entity or a list of entities may be rendered. The transform list may be supplied as nil.

RRVIND - KI function to generate view independent rendering data

Syntax:	(RRVIND int address address tag tag) => (ifail)
Args:	nopts opt_array opt_data entities transforms
Returns:	(ifail)
Syntax:	(rrvind '((token <double>...) ...) '(tag ...) '(tag ...)) => t
Args:	'((option <data> ...) ...) '(entity ...) '(transform ...)
Returns:	t
Example:	(rrvind '((RROPIE) (RROPPA 0.5 0.5)) 7 nil)
Notes:	Some rrvind options are accompanied by real data values. Either a single entity or a list of entities may be rendered. The transform list may be supplied as nil.

SAVMOD - KI function to save model in archive

Syntax:	(SAVMOD tag int string) => (ifail)
Args:	part length name
Returns:	(ifail)

Syntax:	(savmod tag string) => t
Args:	part name
Returns:	t

SAVSNP - KI function to make a snapshot

Syntax:	(SAVSNP int string int) => (ifail)
Args:	length name unused
Returns:	(ifail)
Syntax:	(savsnp string) => t
Args:	name
Returns:	t

SCRIBE - KI function to scribe a line on a face, a region or a wire body

Syntax:	(SCRIBE tag tag vector vector) => (tag int tag int ifail)
Args:	topol curve start end
Returns:	(new_edges nedges new_faces nfaces ifail)
Syntax:	(scribe tag tag vector vector) => ((tag ...) (tag ...))
Args:	topol curve start end
Returns:	((new_edge ...) (new_face...))

SEBBCO - KI function to set bulletin board controls

Syntax:	(SEBBCO int address address int address) => (ifail)
Args:	nentities entity_tokens event_tokens nopts option_tokens
Returns:	(ifail)
Syntax:	(sebbco '(((token ...) token ...) ...) '(token ...))
Args:	'(((entity_type ...) event ...) ...) '(option ..)
Returns:	t
Example:	(sebbco '(((TYTOFA TYTOED) (BBEVCR BBEVDE BBEVCH)) (TYTOBY (BBEVCR BBEVDE))) '(BBOPON)) (setq saved_setting (oubbco)) (sebbco nil '(BBOPOF)) ... (apply 'sebbco saved_setting)

SECTBY - KI function to section bodies

Syntax:	(SECTBY tag tag) => (tag tag tag int ifail)
Args:	bodies surface
Returns:	(front_bodies back_bodies new_faces nfaces ifail)
Syntax:	(sectby tag tag) => ((tag ...) (tag ...) (tag ...))
Args:	bodies surface
Returns:	((front_body...) (back_body...) (new_face ...))

SEINTP - KI function to set interface parameter

Syntax:	(SEINTP int int double) => (ifail)
Args:	option ival rval
Returns:	(ifail)
Syntax:	(seintp token int) => t
Args:	option ival
Returns:	t
Example:	(seintp 'SLIPRB 1000000) (seintp 'SLIPCH t)
Notes:	In the cases where 1 means “on” and 0 means “off”, logical values t and nil may be used in place of integers.

SEMODP - KI function to set modeler parameter

Syntax:	(SEMODP int int double) => (ifail)
Args:	option ival rval
Returns:	(ifail)
Syntax:	(semodp token rval) => t
Args:	option precision
Returns:	t

SESTPA - KI function to set state of part

Syntax:	(SESTPA tag int) => (ifail)
Args:	part state
Returns:	(ifail)
Syntax:	(sestpa tag token) => t

Args:	part state
Returns:	t

SETLEN - KI function to associate a tolerance value with a face, edge or vertex

Syntax:	(SETLEN tag double) => (tag int ifail)
Args:	entity tolerance
Returns:	(new_edges n_edge ifail)
Syntax:	(setlen tag double) => (<tag> ...)
Args:	entity tolerance
Returns:	(<edge> ...)
Example:	(setlen 20 0.001)

SEUFEN - KI function to set user field of entity

Syntax:	(SEUFEN tag address) => (ifail)
Args:	entity workspace_address
Returns:	(ifail)
Syntax:	(seufen tag '(int ...)) => t
Args:	entity int_list
Returns:	t

SHAREN - KI function to share the underlying geometry of a body

Syntax:	(SHAREN tag int address) => (int ifail)
Args:	body nopts opt_array
Returns:	(ngeom ifail)
Syntax:	(sharen tag '(<(token)> ...)) => int
Args:	body '(<(option)> ...)
Returns:	(ngeom)
Example:	(sharen 26 nil) (sharen 26 '((SHOPIC)))

SIMPEN - KI function to simplify geometry in a body

Syntax:	(SIMPEN tag int) => (tag int ifail)
Args:	body level

Returns:	(geometry ngeometry ifail)
Syntax:	(simpen tag token) => (tag ...)
Args:	body level
Returns:	(new_geometry ...)

SPLTEN - KI function to split topology and geometry of body at any G1 discontinuities

Syntax:	(SPLTEN tag) => (int tag tag ifail)
Args:	body
Returns:	(n_splt_faces faces_split new_faces ifail)
Syntax:	(splten tag) => (<(tag (tag ...))> ...)
Args:	body
Returns:	(<(old_face (new_face ...)) > ...)
Example:	(splten 26)

SRCHIL - KI function to search for a value in a list of integers from a starting index

Syntax:	(SRCHIL tag int int) => (int ifail)
Args:	list value start
Returns:	(index ifail)
Syntax:	(srchil tag int <int>) => int
Args:	list value <start . 1>
Returns:	index

SRCHRL - KI function to search for a value in a list of reals from a starting index

Syntax:	(SRCHRL tag double int) => (int ifail)
Args:	list value start
Returns:	(index ifail)
Syntax:	(srchrl tag double <int>) => int
Args:	list value <start . 1>
Returns:	index

SRCHTG - KI function to search for a value in a list of tags from a starting index

Syntax:	(SRCHTG tag tag int) => (int ifail)
Args:	list value start
Returns:	(index ifail)
Syntax:	(srchtg tag tag <int>) => int
Args:	list value <start . 1>
Returns:	index

STAMOD - KI function to start the modeler

Syntax:	(STAMOD logical int string int) => (tag int ifail)
Args:	kijon nchars journal_filename user_field
Returns:	(world ki_version ifail)
Syntax:	(stamod <string> <int>) => string
Args:	<journal_filename . “”> <user_field . 0>
Returns:	ki_version
Example:	(stamod) (stamod “my_journal”) (stamod 'op 4)
Notes:	The default journalling is off, the default user field length is 0. The ifail KI_modeller_not_stopped is not treated as an error, should it occur this string is returned instead of the version number string.

STOMOD - KI function to stop modeler

Syntax:	(STOMOD) => (ifail)
Args:	-none-
Returns:	(ifail)
Syntax:	(stomod) => string
Args:	-none-
Returns:	ifail_mnemonic
Notes:	The ifail KI_modeller_not_started is not treated as an error. This string is returned instead of KI_no_errors where necessary.

SUBBYS - KI function to subtract bodies

Syntax:	(SUBBYS tag tag) => (tag int ifail)
Args:	target tool
Returns:	(assembly nbodies ifail)
Syntax:	(subbys tag '(tag ...)) => (tag int)
Args:	target '(tool ...)
Returns:	(assembly nbodies)
Example:	(subbys 7 71) (subbys 7 '(71 145))

SWEENT - KI function to sweep entity

Syntax:	(SWEENT tag vector) => (tag int int ifail)
Args:	entity path
Returns:	(laterals nlaterals state ifail)
Syntax:	(sweent '(tag ...) vector) => ((tag ...) token)
Args:	'(entity ...)
Returns:	((new_lateral ...) state)
Example:	(sweent '(6 10 22) '(0 0 1)) (sweent 7 '(5 5 5))
Notes:	sweent can be applied to a single body, vertex or face, or a list of faces.

SWIENT - KI function to swing entity

Syntax:	(SWIENT tag vector vector double) => (tag int int ifail)
Args:	entity point direction angle
Returns:	(new_laterals nlaterals state ifail)
Syntax:	(swient '(tag ...) vector vector double) => ((tag ...) state)
Args:	'(entity ...) point direction angle
Returns:	((new_lateral ...) state)
Example:	(swient 7 '(0 0 0) '(0 0 1) 1.57) (swient '(19 64) '(0 0 0) '(0 0 1) pi_2)
Notes:	swient accepts either a single vertex, face or body or a list of faces.

TAPFAS - KI function to taper faces in a body

Syntax:	(TAPFAS tag vector vector double) => (tag int int ifail)
Args:	faces point direction angle
Returns:	(tapered_faces nfaces state ifail)
Syntax:	(tapfas '(tag ...) vector vector double) => ((tag ...) token)
Args:	'(face ...) point direction angle
Returns:	((tapered_face ...) state)
Example:	(tapfas '(6 38 100) '(0 0 0) '(0 0 1) 0.3)

THIKEN - KI function to thicken a sheet body into a solid

Syntax:	(THIKEN tag double double logical double int) => (tag tag tag int ifail)
Args:	sheet front_thickness back_thickness check tolerance max_flts
Returns:	(old_topol new_topol bad_topol state ifail)
Syntax:	(thiken tag double double logical double int) => ((tag ...) (tag ...) (<tag> ...) token)
Args:	sheet front_thickness back_thickness check tolerance max_flts
Returns:	((old _topol ...) (new_topol ...) (<bad_topol> ...) state)
Example:	(thiken 146 .1 t .0000001 0)

TRIMSH - KI function to trim a sheet body to supplied curves

Syntax:	(TRIMSH tag int address int address address) => (ifail)
Args:	sheet n_curves curves n_opts opt_array opt_data
Returns:	(ifail)
Syntax:	(trimsh tag (tag ...) <((token double ...) ...)>) => t
Args:	sheet (curve ...) <'((option data) ...))>
Returns:	t
Example:	(trimsh 21 '(34 35 36 37) '((SLTRRE 1.0 1.0 1.0 3.0 3.0 3.0)))
Notes:	(i) At present the only options available are SLTRRE and SLTRKE. These are mutually exclusive and one of them must be supplied. (ii) The option data may be bracketed to aid readability, i.e. '((SLTRRE 1.0 1.0 1.0 3.0 3.0 3.0)) and '((SLTRRE (1.0 1.0 1.0) (3.0 3.0 3.0))) are equivalent.

TRSHCU - KI function to trim a sheet body with curves

Syntax:	(TRSHCU tag int address int address vector) => (tag int tag tag ifail)
Args:	sheet ncurves trim_curves nopts opts direction
Returns:	(edges nedges derived_curves original_edges ifail)
Syntax:	(trshcu tag '(tag ...) '((token) ...) vector) => ((tag ...) (tag ...) (logical ...))
Args:	sheet trim_curves options direction
Returns:	((edge ...) (derived_curve ...) (original_edge ...))
Example:	(trshcu 21 '(33) '(TRSHPD) '(0 0 1))

TWEFAC - KI function to transform geometry of faces

Syntax:	(TWEFAC tag tag) => (int ifail)
Args:	faces transforms
Returns:	(state ifail)
Syntax:	(twefac '(tag ...) '(tag ...)) => token
Args:	faces transforms
Returns:	state
Example:	(twefac 100 150) (twefac '(64 100) 150) (twefac '(36 (64 100)) '(150 170))

TWSUFA - KI function to tweak the surface(s) of face(s)

Syntax:	(TWSUFA tag tag logical) => (int ifail)
Args:	face surface sense
Returns:	(state ifail)
Syntax:	(twsufa '(tag ...) '(tag ...) `(logical ...) => token
Args:	faces surfaces senses
Returns:	state
Example:	(twsufa 22 35 t) (twsufa '(22 25) '(35 38) '(t f)

UNIBYS - KI function to unite bodies

Syntax:	(UNIBYS tag tag) => (tag int ifail)
Args:	target tools
Returns:	(assembly nbodies ifail)

Syntax:	(unibys tag '(tag ...)) => (tag int)
Args:	target '(tool ...)
Example:	(unibys 7 45) (unibys 7 '(130 212))
Returns:	(assembly nbodies)

UNLDPA - KI function to unload part

Syntax:	(UNLDPA tag) => (ifail)
Args:	part
Returns:	(ifail)
Syntax:	(unldpa tag) => t
Args:	(unldpa part)
Returns:	t

Special Kernel Interface Routines

E

E.1 Introduction

This chapter contains the specifications of all the Special Kernel Interface routines in alphabetical order. These special routines are supplied only to provide access to pre-v9 behaviour in their specific area of functionality. Their use is not encouraged other than in the particular instance as they will be withdrawn in a future version.

> **Warning:** Do not use these functions unless you know exactly why you need them and the equivalent standard routine does not provide the desired functionality, the correct way to obtain it is through the PK.

BOPOLD - Global or local boolean operation on bodies

Receives

```
KI_tag_list_entity   *target         target body or list of faces
KI_tag_list_entity   *tools          tool bodies or faces of tool body
<KI_int_nitems>      *nopts          number of boolean options
KI_cod_boop           opts[nopts]    boolean option codes
<KI_tag_list_entity>  optdta[nopts]  boolean option data lists
```

Returns

```
KI_tag_list_body     *bodys          resulting bodies
KI_int_nitems        *nbodys         number of bodies
KI_cod_error         *ifail          failure code
```

Specific Errors

KI_partial_no_intersect	No imprinting in local boolean
KI_boolean_failure	Unrecognized result; Invalid matched region; Inconsistent arguments, or internal error
KI_non_manifold	Non-manifold result
KI_t_sheet	T-sheet
KI_partial_coi_found	Boolean failure due to partial coincidence
KI_cant_intsc_solid_sheet	Cant intersect solid target with sheet tool bodies
KI_solid_has_void	Illegal void
KI_not_solid	Body is not solid
KI_not_sheet	Body is not sheet
KI_opposed_sheets	Attempt to unite opposed sheets
KI_cant_unite_solid_sheet	Attempt to unite solid and sheet
KI_unsuitable_topology	A faceset selector is from boundary or wrong body
KI_tool_faces_many_bodies	Tool faces are from more than one body

KI_targ_faces_many_bodies	Target faces are from more than one body
KI_mixed_sheets_solids	Mixture of sheet and solid tool bodies
KI_instanced_tools	Instanced tool bodies
KI_duplicate_targets	Duplication in list of target faces
KI_duplicate_tools	Duplication in list of tool bodies
KI_too_many_targets	Too many target bodies
KI_unsuitable_entity	Target or tool not a face or body
KI_wire_body	Target or tool has wireframe or acorn components
KI_missing_geom	Target or tool has incomplete geometry
KI_same_tool_and_target	Tool body is also target body
KI_contradictory_request	Bad combination of options or data for type of boolean
KI_not_in_same_partition	Target and tools are not all in the same partition
KI_general_body	Target or tool is general body

Description This function takes the same arguments as BOPBYS. Its only purpose is to provide access to pre-v9 behaviour in partial booleans with excluded faces. As such, its use is STRONGLY DEPRECATED and it will be withdrawn in a future version. Do not use this unless you know exactly why you need it: if BOPBYS does not provide the desired functionality, the correct way to obtain it is through the PK.

Index In

L

M

N

O

P

R

S

T

U

V

W