Parasolid V13.0
PK Interface Programming Reference Manual

Part 2 – Interface structures, Tokens, and Error Codes
June 2001

Important Note

This Software and Related Documentation are proprietary to Unigraphics Solutions Inc.

© Copyright 2001 Unigraphics Solutions Inc. All rights reserved

Restricted Rights Legend: This commercial computer software and related documentation are provided with restricted rights. Use, duplication or disclosure by the U.S. Government is subject to the protections and restrictions as set forth in the Unigraphics Solutions Inc. commercial license for the software and/or documentation as prescribed in DOD FAR 227-7202-3(a), or for Civilian agencies, in FAR 27.404(b)(2)(i), and any successor or similar regulation, as applicable. Unigraphics Solutions Inc. 10824 Hope Street, Cypress, CA 90630

This documentation is provided under license from Unigraphics Solutions Inc. This documentation is, and shall remain, the exclusive property of Unigraphics Solutions Inc. Its use is governed by the terms of the applicable license agreement. Any copying of this documentation, except as permitted in the applicable license agreement, is expressly prohibited.

The information contained in this document is subject to change without notice and should not be construed as a commitment by Unigraphics Solutions Inc. who assume no responsibility for any errors or omissions that may appear in this documentation.

Unigraphics Solutions™

Parker's House
46 Regent Street
Cambridge CB2 1DP
UK
Tel: +44 (0)1223 371555
Fax: +44 (0)1223 316931
email: ps-support@ugs.com
Web: www.parasolid.com

Trademarks

Parasolid is a trademark of Unigraphics Solutions Inc.

HP and HP-UX are registered trademarks of Hewlett-Packard Co.

SPARCstation and Solaris are trademarks of Sun Microsystems, Inc.

Alpha AXP and VMS are trademarks of Digital Equipment Corp.

IBM, RISC System/6000 and AIX are trademarks of International Business Machines Corp.

OSF is a registered trademark of Open Software Foundation, Inc.

UNIX is a registered trademark of UNIX Systems Laboratories, Inc.

Microsoft Visual C/C++ and Window NT are registered trademarks of Microsoft Corp.

Intel is a registered trademark of Intel Corp.

Silicon Graphics is a registered trademark, and IRIX a trademark, of Silicon Graphics, Inc.

Table of Contents

1 PK Interface Structures 5

1.1 Introduction **5**
1.2 PK interface structure list **5**

A PK Tokens (Numeric) 295

A.1 Numerical Listing of PK Interface Tokens **295**

B PK Tokens (Alphabetic) 325

B.1 Alphabetical List of PK Interface Tokens **325**

C PK Error Codes (Numeric) 369

C.1 Numerical List of PK Error Codes **369**
C.1.1 Error Codes 0-99 **369**
C.1.2 Error Codes 100-999 **371**
C.1.3 Error Codes 1000-4999 **376**
C.1.4 Error Codes 5000-9999 **379**

D PK Error Codes (Alphabetic) 385

D.1 Alphabetical List of PK Error Codes **385**
D.1.1 Error Codes FG* **385**
D.1.2 Error Codes a-b **385**
D.1.3 Error Codes c-f **389**
D.1.4 Error Codes g-o **393**
D.1.5 Error Codes p-z **396**

PK Interface Structures 1

1.1 Introduction

This chapter contains the specifications of all the PK Interface **structures** in alphabetical order.

The specifications of the PK Interface **functions** are in Chapter 2, “PK Interface Functions” of the *PK Interface Programming Reference Manual* (Part 1 - Functions).

1.2 PK interface structure list

PK_abort_reason_t

```
typedef int PK_abort_reason_t;
```

This type is used by an application to specify a reason for requesting Parasolid to abort a modelling operation.

PK_abort_reason_t	Meaning
PK_abort_user_interrupt_c	Abort requested to handle user interrupt
PK_abort_runtime_error_c	Abort requested to handle runtime error
PK_abort_frustrum_error_c	Abort requested to handle error in frustrum

PK_ASSEMBLY_array_t

This structure contains an array of entities of type PK_ASSEMBLY_t

```
typedef struct PK_ASSEMBLY_array_s
    {
    PK_ASSEMBLY_t  *array;   --- array of ASSEMBLYs
    int             length;  --- length of array
    }
    PK_ASSEMBLY_array_t;
```

It has the fields:

'array': array of entities of type PK_ASSEMBLY_t

'length': the number of entities in 'array'

PK_ASSEMBLY_t

This type represents an ASSEMBLY. An assembly is a collection of instances.

An assembly is a type of part (PK_CLASS_assembly is a subclass of PK_CLASS_part).

PK_ATTDEF_array_t

This structure contains an array of entities of type PK_ATTDEF_t

```
typedef struct PK_ATTDEF_array_s
    {
    PK_ATTDEF_t    *array;   --- array of ATTribute DEFinitions
    int             length;  --- length of array
    }
    PK_ATTDEF_array_t;
```

It has the fields:

'array': array of entities of type PK_ATTDEF_t

'length': the number of entities in 'array'

PK_ATTDEF_callback_flags_t

```
typedef struct PK_ATTDEF_callback_flags_s
    {
    PK_LOGICAL_t split_callback_on;
    PK_LOGICAL_t merge_callback_on;
    PK_LOGICAL_t delete_callback_on;
    PK_LOGICAL_t copy_callback_on;
    PK_LOGICAL_t transmit_callback_on;
    PK_LOGICAL_t receive_callback_on;
    }
    PK_ATTDEF_callback_flags_t;
```

This data structure holds logical flags which are set using PK_ATTDEF_set_callback_flags. An attribute callback function will only be called if it is not NULL, and its corresponding flag is set to PK_LOGICAL_true.

PK_ATTDEF_callback_fns_t

```
typedef void (*PK_ATTDEF_split_callback_f_t)
    ( PK_ENTITY_t old_entity,
      int n_attribs, const PK_ATTRIB_t attribs[],
      PK_ENTITY_t new_entity );

typedef void (*PK_ATTDEF_merge_callback_f_t)
    ( PK_ENTITY_t live_entity,
      int n_live_attribs, const PK_ATTRIB_t live_attribs[],
      PK_ENTITY_t doomed_entity,
      int n_doomed_attribs, const PK_ATTRIB_t doomed_attribs[] );

typedef void (*PK_ATTDEF_delete_callback_f_t)
    ( PK_ENTITY_t entity,
      int n_attribs, const PK_ATTRIB_t attribs[] );

typedef void (*PK_ATTDEF_copy_callback_f_t)
    ( PK_ENTITY_t old_entity,
      int n_attribs, const PK_ATTRIB_t attribs[],
      PK_ENTITY_t new_entity );

typedef void (*PK_ATTDEF_transmit_callback_f_t)
    ( PK_ENTITY_t entity,
      int n_attribs, const PK_ATTRIB_t attribs[] );

typedef void (*PK_ATTDEF_receive_callback_f_t)
    ( PK_ENTITY_t entity,
      int n_attribs, const PK_ATTRIB_t attribs[] );

typedef struct PK_ATTDEF_callback_fns_s
    {
    PK_ATTDEF_split_callback_f_t    split_fn;
    PK_ATTDEF_merge_callback_f_t    merge_fn;
    PK_ATTDEF_delete_callback_f_t   delete_fn;
    PK_ATTDEF_copy_callback_f_t     copy_fn;
    PK_ATTDEF_transmit_callback_f_t transmit_fn;
    PK_ATTDEF_receive_callback_f_t  receive_fn;
    }
    PK_ATTDEF_callback_fns_t;
```

This data structure holds pointers to the functions the application wishes to register with the PK using PK_ATTDEF_register_callbacks or PK_ATTDEF_register_cb. Any of these may be NULL.

PK_ATTDEF_callback_type_t

```
typedef int PK_ATTDEF_callback_type_t;
```

This data type specifies the type of an attribute callback function. It has values:

PK_ATTDEF_callback_normal_c	Normal callback - application processes the attributes.
PK_ATTDEF_callback_read_only_c	Read-only callback - Parasolid processes the attributes.

PK_ATTDEF_class_t

```
typedef int PK_ATTDEF_class_t;
```

PK_ATTDEF_class_01_c	attribute is independent of physical size and position of entity to which it is attached (e.g. density)
PK_ATTDEF_class_02_c	attribute is dependent on entity size but not on position (e.g. weight)
PK_ATTDEF_class_03_c	attrubute may vary with position or orientation (e.g. moment of inertia)
PK_ATTDEF_class_04_c	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)
PK_ATTDEF_class_05_c	attribute transforms with its owner provided its owner is not changed in other ways (e.g. centre of gravity)
PK_ATTDEF_class_06_c	as for PK_ATTDEF_class_01_c, 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
PK_ATTDEF_class_07_c	as for PK_ATTDEF_class_04_c, attribute transforms with its owner, but is otherwise independent of size and shape of the owner. However this class supports multiple values - an entity may have a list of attributes of the same type attached.

PK_ATTDEF_register_cb_o_t

```
struct PK_ATTDEF_register_cb_o_s
    {
    int                         o_t_version;
                                    --- version number of option structure
    PK_ATTDEF_callback_type_t callback_type;   --- normal or read_only
    };

typedef struct PK_ATTDEF_register_cb_o_s
               PK_ATTDEF_register_cb_o_t;
```

Holds optional controls for registering a callback

The option structure defines options applicable to registering an attribute callback.

Description of fields:

'callback_type'	Type of callback function. Permitted values are: PK_ATTDEF_callback_normal_c (the default) PK_ATTDEF_callback_read_only_c

PK_ATTDEF_sf_2_t

```
struct PK_ATTDEF_sf_2
    {
    char                *name;              --- name of attribute type
    PK_ATTDEF_class_t    attdef_class;      --- transform characteristics
    int                  n_owner_types;     --- number of legal owners (>0)
    PK_CLASS_t          *owner_types;       --- legal owner types
    int                  n_fields;          --- number of fields (>= 0)
    PK_ATTRIB_field_t   *field_types;       --- types of above fields
```

```
    PK_LOGICAL_t         unicode_names;    --- Unicode or char names
    PK_field_names_t     field_names;      --- names of above fields
    int                 *field_sizes;      --- for future use
    };
typedef struct PK_ATTDEF_sf_2 PK_ATTDEF_sf_2_t;
```

This data structure is the standard form for a named-field attribute definition.

Description of fields:

'name'

Acts as a label for the type which is fixed across transmit and receive. It is specified by a string containing only printable characters. Names beginning with the string SDL/TY are reserved for attribute types generated internally by Parasolid.

'attdef_class'

Determines the behaviour of an attribute of the new type when the entity to which the attribute is attached is changed by a modelling operation. The meaning of each possible value of 'class' is as follows.

PK_ATTDEF_class_01_c	Attribute is independent of physical size and position of entity to which it is attached (e.g. density).
PK_ATTDEF_class_02_c	Attribute is dependent on entity size but not on position. (e.g. weight).
PK_ATTDEF_class_03_c	Attribute may vary with position or orientation (e.g. moment of inertia).
PK_ATTDEF_class_04_c	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)
PK_ATTDEF_class_05_c	Attribute transforms with its owner provided its owner is not changed in other ways (e.g. centre of gravity)
PK_ATTDEF_class_06_c	As for class 01, 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.
PK_ATTDEF_class_07_c	As for class 04, attribute transforms with its owner, but is otherwise independent of size and shape of the owner. However this class supports multiple values - an entity 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.

axis fields behave as a coordinate field and a direction field.

'owner_types'
An array of length 'n_owner_types' which dictates the entity class which may legitimately own attributes of the given type. Legitimate classes are as follows.

```
PK_CLASS_assembly
PK_CLASS_instance
PK_CLASS_body
PK_CLASS_region
PK_CLASS_shell
PK_CLASS_face
PK_CLASS_loop
PK_CLASS_edge
PK_CLASS_fin
PK_CLASS_vertex
PK_CLASS_group
PK_CLASS_surf
PK_CLASS_curve
PK_CLASS_point
```

The geometric entity classes, PK_CLASS_surf, PK_CLASS_curve and PK_CLASS_point, may only be specified as legal owners for attribute types with class PK_ATTDEF_class_01_c or PK_ATTDEF_class_04_c.

A valid attribute definition must specify at least one owner.

'field_types'
An array of length 'n_fields' which specifies the sequence and type of fields for attributes of the given type. The permitted field types are as follows.

```
PK_ATTRIB_field_real_c
PK_ATTRIB_field_integer_c
PK_ATTRIB_field_string_c
PK_ATTRIB_field_ustring_c
PK_ATTRIB_field_vector_c
PK_ATTRIB_field_coordinate_c
PK_ATTRIB_field_direction_c
PK_ATTRIB_field_axis_c
PK_ATTRIB_field_pointer_c
```

'n_fields' may be zero.

'unicode_names'
A switch specifying whether names are supplied in Unicode form, or in the native character set of the machine.

Currently, only native names are supported, so PK_LOGICAL_false is required.

'field_names'
An array of length 'n_fields' which specifies the names of fields for attributes of the given type.

'field_sizes'
For future use.

Currently this field must be set to NULL.

PK_ATTDEF_sf_t

```
struct PK_ATTDEF_sf
    {
    char                *name;              --- name of attribute type
    PK_ATTDEF_class_t    attdef_class;      --- transform characteristics
    int                  n_owner_types;     --- number of legal owners (>0)
    PK_CLASS_t          *owner_types;       --- legal owner types
    int                  n_fields;          --- number of fields (>= 0)
    PK_ATTRIB_field_t   *field_types;       --- types of above fields
    };
typedef struct PK_ATTDEF_sf PK_ATTDEF_sf_t;
```

This data structure is the standard form for an attribute definition.

Description of fields:

'name'
Acts as a label for the type which is fixed across transmit and receive. It is specified by a string containing only printable characters. Names beginning with the string SDL/TY are reserved for attribute types generated internally by Parasolid.

'attdef_class'
Determines the behaviour of an attribute of the new type when the entity to which the attribute is attached is changed by a modelling operation. The meaning of each possible value of 'class' is as follows.

PK_ATTDEF_class_01_c	Attribute is independent of physical size and position of entity to which it is attached (e.g. density).
PK_ATTDEF_class_02_c	Attribute is dependent on entity size but not on position. (e.g. weight).
PK_ATTDEF_class_03_c	Attribute may vary with position or orientation (e.g. moment of inertia).
PK_ATTDEF_class_04_c	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)
PK_ATTDEF_class_05_c	Attribute transforms with its owner provided its owner is not changed in other ways (e.g. centre of gravity)
PK_ATTDEF_class_06_c	As for class 01, 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.
PK_ATTDEF_class_07_c	As for class 04, attribute transforms with its owner, but is otherwise independent of size and shape of the owner. However this class supports multiple values - an entity 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.

axis fields behave as a coordinate field and a direction field.

'owner_types' An array of length 'n_owner_types' which dictates the entity class which may legitimately own attributes of the given type. Legitimate classes are as follows.

```
PK_CLASS_assembly
PK_CLASS_instance
PK_CLASS_body
PK_CLASS_region
PK_CLASS_shell
PK_CLASS_face
PK_CLASS_loop
PK_CLASS_edge
PK_CLASS_fin
PK_CLASS_vertex
PK_CLASS_group
PK_CLASS_surf
PK_CLASS_curve
PK_CLASS_point
```

The geometric entity classes, PK_CLASS_surf, PK_CLASS_curve and PK_CLASS_point, may only be specified as legal owners for attribute types with class PK_ATTDEF_class_01_c or PK_ATTDEF_class_04_c.

A valid attribute definition must specify at least one owner.

'field_types' An array of length 'n_fields' which specifies the sequence and type of fields for attributes of the given type. The permitted field types are as follows.

```
PK_ATTRIB_field_real_c
PK_ATTRIB_field_integer_c
PK_ATTRIB_field_string_c
PK_ATTRIB_field_ustring_c
PK_ATTRIB_field_vector_c
PK_ATTRIB_field_coordinate_c
PK_ATTRIB_field_direction_c
PK_ATTRIB_field_axis_c
PK_ATTRIB_field_pointer_c
```

'n_fields' may be zero.

PK_ATTDEF_t

This type represents an ATTribute DEFinition. An attribute definition determines what information an attribute may contain and how the attribute behaves during various modelling operations.

An attribute definition is a type of entity (PK_CLASS_attdef is a subclass of PK_CLASS_entity).

PK_ATTRIB_array_t

This structure contains an array of entities of type PK_ATTRIB_t

```
typedef struct PK_ATTRIB_array_s
    {
    PK_ATTRIB_t    *array;   --- array of ATTRIButes
    int             length;  --- length of array
    }
    PK_ATTRIB_array_t;
```

It has the fields:

'array': array of entities of type PK_ATTRIB_t

'length': the number of entities in 'array'

PK_ATTRIB_field_t

typedef int PK_ATTRIB_field_t;

```
    PK_ATTRIB_field_integer_c
    PK_ATTRIB_field_real_c
    PK_ATTRIB_field_string_c
    PK_ATTRIB_field_ustring_c
    PK_ATTRIB_field_vector_c
    PK_ATTRIB_field_coordinate_c
    PK_ATTRIB_field_direction_c
    PK_ATTRIB_field_axis_c
    PK_ATTRIB_field_entity_c
    PK_ATTRIB_field_pointer_c
```

PK_ATTRIB_filter_f_t

typedef PK_LOGICAL_t (*PK_ATTRIB_filter_f_t)(const int *, void *);

This is the function pointer type to be passed to PK_PART_ask_attribs_filter and PK_PART_ask_attrib_owners.

The function receives two arguments: an array of integers containing selected integer field values of an attribute, and a pointer to (optional) context information supplied by the application to the PK function. It returns true or false depending on whether the values match the application's criteria.

PK_ATTRIB_t

This type represents an ATTRIBute. An attribute is attached to an entity and contains additional information which is frequently application defined.

An attribute is a type of entity (PK_CLASS_attrib is a subclass of PK_CLASS_entity).

What information the attribute may contain, and how the attribute behaves during various modelling operations is defined by an attribute definition (PK_ATTDEF_t).

PK_AXIS1_sf_t

```
struct PK_AXIS1_sf_s
    {
    PK_VECTOR_t  location;   --- a cartesian point.
    PK_VECTOR1_t axis;       --- a direction.
    };
typedef struct PK_AXIS1_sf_s PK_AXIS1_sf_t;
```

This data structure is the standard form for a single axis.

This corresponds to STEP's axis1_placement structure.

PK_AXIS2_sf_t

```
struct PK_AXIS2_sf_s
    {
    PK_VECTOR_t  location;          --- a cartesian point.
    PK_VECTOR1_t axis;              --- a direction.
    PK_VECTOR1_t ref_direction;     --- a direction.
    };
typedef struct PK_AXIS2_sf_s PK_AXIS2_sf_t;
```

This data structure is the standard form for the location and orientation of two mutually perpendicular axes.

Specific Errors

PK_ERROR_vectors_not_orthogonal 'axis' and 'ref_direction' are not othogonal

This corresponds to STEP's axis2_placement structure.

PK_BB_event_t

```
typedef int PK_BB_event_t;
```

This type has the values:

PK_BB_event_create_c	entity created
PK_BB_event_delete_c	entity deleted
PK_BB_event_copy_c	entity copied
PK_BB_event_transfer_c	entities transferred
PK_BB_event_merged_c	entities merged
PK_BB_event_split_c	entities split
PK_BB_event_transform_c	entity transformed
PK_BB_event_change_attrib_c	attrib owned by entity changed
PK_BB_event_change_c	entity changed

PK_BB_sf_t

```
struct PK_BB_sf_s
    {
    PK_CLASS_array_t create;        --- creation classes
    PK_CLASS_array_t deleet;        --- deletion classes
    PK_CLASS_array_t copy;          --- copy classes
    PK_CLASS_array_t transfer;      --- transfer classes
```

```
        PK_CLASS_array_t merge;         --- merge classes
        PK_CLASS_array_t split;         --- split classes
        PK_CLASS_array_t transform;     --- transform classes
        PK_CLASS_array_t change_attrib; --- attribute classes
        PK_CLASS_array_t change;        --- change classes
        };
    typedef struct PK_BB_sf_s PK_BB_sf_t;
```

This standard form defines a bulletin board.

Specific Errors

PK_ERROR_bad_class_event_comb class not allowed for event

The fields determine, for the various possible events, which classes of entities will lead to events being recorded in the bulletin board.

The allowed combinations of classes and events are indicated by crosses in the following table:

		create	delete	copy	transfer	merge	split	transform	change_attrib	change
PK_CLASS_part	PK_CLASS_body	X	X	X		X	X	X	X	X
	PK_CLASS_assembly	X	X	X		X	X	X	X	
PK_CLASS_topol	PK_CLASS_instance	X	X	X	X	X	X	X	X	X
	PK_CLASS_shell	X	X	X	X	X	X	X	X	X
	PK_CLASS_face	X	X	X	X	X	X	X	X	X
	PK_CLASS_loop	X	X	X	X	X	X	X	X	X
	PK_CLASS_edge	X	X	X	X	X	X	X	X	X
	PK_CLASS_vertex	X	X	X	X	X	X	X	X	X
	PK_CLASS_region	X	X	X	X	X	X	X	X	X
PK_CLASS_geom	PK_CLASS_surf	X	X	X	X			X	X	X
	PK_CLASS_curve	X	X	X	X			X	X	X
	PK_CLASS_point	X	X	X	X			X	X	X
	PK_CLASS_group	X	X	X	X	X	X	X	X	X
	PK_CLASS_transf	X	X	X	X			X	X	X
	PK_CLASS_attdef	X								

A BB standard form given to PK_BB_create may specify the superclasses PK_CLASS_part, PK_CLASS_topol or PK_CLASS_geom where bulletining is required for all of their subclasses and all of those subclasses are allowed for the event. BB standard forms returned by PK_BB_ask do not use these superclasses (they are expanded into their subclasses).

Note that the name of the field for delete classes is 'deleet'. This is because 'delete' is a reserved word in C++.

PK_BB_status_t

typedef int PK_BB_status_t;

This type has the values:

PK_BB_status_off_c	bulletin board off and clear contents
PK_BB_status_on_c	bulletin board on to record entities
PK_BB_status_user_field_c	bulletin board on to record entities and user fields.

PK_BCURVE_array_t

This structure contains an array of entities of type PK_BCURVE_t

```
typedef struct PK_BCURVE_array_s
    {
    PK_BCURVE_t    *array;   --- array of B geometry CURVEs
    int             length;  --- length of array
    }
    PK_BCURVE_array_t;
```

It has the fields:

'array': array of entities of type PK_BCURVE_t

'length': the number of entities in 'array'

PK_BCURVE_create_by_fitting_o_t

```
struct PK_BCURVE_create_by_fitting_o_s
{
int                          o_t_version; --- option structure version number
int                          n_curves;    --- number of bcurves to create (0)
PK_BCURVE_fit_data_t         *fit_data;   --- array of fitting data (NULL)
PK_INTERVAL_t                range;       --- parameter range for bcurves
                                              --- (0,1)
double                       tolerance;   --- distance tolerance (0.00001)
double                       angular_tolerance; --- angular tolerance (0.0)
int                          n_break_parms;     --- number of break parms (0)
double                       *break_parms;      --- break-parameters (NULL)
};

typedef struct PK_BCURVE_create_by_fitting_o_s PK_BCURVE_create_by_fitting_o_t;
```

Holds optional controls for the fitting of bcurves.

This structure contains the following fields:

'n_curves'	the number of bcurves to be produced.
'fit_data'	an array of length 'n_curves' specifying for each bcurve the fitting method to be used and the associated data.
'range'	the parameter range for the resultant bcurves. The bcurves will exactly fit the sample points at the start and end of this range.
'tolerance'	the distance tolerance to within which the bcurves will fit the sample data.

'angular_tolerance' if non-zero, the maximum angle between the tangents of the bcurves and the tangents from the sample data. Note that this will require the calculation of first derivatives at all sample parameters during the fit.

'n_break_parms' the number of 'break_parms' supplied.

'break_parms' a strictly increasing array of length 'n_break_parms' of parameter values lying within 'range'. The resultant bcurves will exactly pass through the sample points at these values.

PK_BCURVE_fit_chain_t

```
struct PK_BCURVE_fit_chain_s
    {
    int              n_curves_in_chain; --- number of curves in this chain (0)
    PK_CURVE_t      *curves_in_chain;   --- curves in the chain (NULL)
    PK_INTERVAL_t   *curve_intervals;   --- parameter intervals for each
                                        --- chain-curve (NULL)
    double          *fit_parameters;    --- parameters for bcurve at ends
                                        --- of each chain-curve (NULL)
    };

typedef struct PK_BCURVE_fit_chain_s PK_BCURVE_fit_chain_t;
```

This structure contains the data to define a "chain" for the chain-of-curves evaluator function.

This structure contains the following fields:

'n_curves_in_chain' The number of curves in the chain.

'curves_in_chain' An ordered array of 'n_curves_in_chain' curves.

'curve_intervals' A parallel array of parameter intervals on the curves denoting the sections of the curves that make up the chain. Together with the 'curves_in_chain' field this specifies a chain of trimmed curves.

'fit_parameters' A strictly increasing array of 'n_curves_in_chain' + 1 doubles representing the parameters for the fitting bcurves at the end points of the trimmed curves. If the curve-chain is periodic then the last value of 'fit_parameters' should be equal to the first plus the required period.

A chain is a connected set of trimmed curves defined by an ordered array of curves and a matching array of parameter intervals. The curves should meet each other with G1 continuity at the points specified by the parameter intervals. Each curve should be oriented so that its parameter increases in the forward direction along the chain.

The parameterisation of the fitted bcurve with respect to the chain is defined by the 'fit_parameters' field. The fit-parameters specify the parameter values taken by the fitted bcurve at the ends of each of the trimmed curves. Evaluation at the intermediate parameters on the bcurve is by cubic interpolation of the chain-curve parameterisation between the fit-parameters. If a set of break-parameters is provided to PK_BCURVE_create_by_fitting then each of these break-parameters must lie at the end of a trimmed curve and the corresponding fit-parameter must equal the break-parameter.

PK_BCURVE_fit_data_t

```
struct PK_BCURVE_fit_data_s
    {
    PK_BCURVE_fit_eval_type_t    eval_type; --- what evaluator to use
                                            --- (PK_BCURVE_fit_eval_chain_c)
    PK_BCURVE_fit_eval_f_t       eval_fn;   --- user evaluator (NULL)
    PK_BCURVE_fit_eval_data_t    eval_data; --- evaluator data
    PK_BCURVE_fit_err_method_t   err_method;--- error method for bcurve
                                            --- (PK_BCURVE_fit_err_parm_c)
    PK_LOGICAL_t                 rational;  --- if bcurve is to be rational
                                            --- (PK_LOGICAL_false)
    };

typedef struct PK_BCURVE_fit_data_s  PK_BCURVE_fit_data_t;
```

This structure contains the controls for fitting an individual bcurve.

This structure contains the following fields:

'eval_type' This specifies how sample data should be obtained for this bcurve. If 'PK_BCURVE_fit_eval_user_c' is chosen then an evaluator function of the form PK_BCURVE_fit_eval_f_t should be supplied in 'eval_fn'.

'eval_fn' This field allows an application to supply an evaluator function of form PK_BCURVE_fit_eval_f_t.

'eval_data' This contains the data required for the chosen evaluator function. For internal evaluators the application must supply the appropriate associated data. For external (application-supplied) evaluators the application can provide data in this structure which will be passed to the evaluator function whenever Parasolid calls it. Such data will not be examined by Parasolid.

'err_method' This controls how Parasolid measures the error between the bcurve and the sample data. The value 'PK_BCURVE_fit_err_none_c' specifies that the bcurve has no tolerance checking performed on it. Note that there must be at least one 'PK_BCURVE_fit_data_t' structure supplied which does not have this field set to 'PK_BCURVE_fit_err_none_c'.

'rational' This would specify whether the bcurve should be rational. At present this option is not operational and is for future enhancements. It should be left at its default value.

PK_BCURVE_fit_err_method_t

```
typedef int PK_BCURVE_fit_err_method_t;
```

This data type selects how points in the sample data are matched to the approximating bcurve for calculating the error in the fit. It has values:

PK_BCURVE_fit_err_none_c	don't calculate any errors on bcurve
PK_BCURVE_fit_err_parm_c	match sample point to point on bcurve with equal parameter
PK_BCURVE_fit_err_closest_c	match sample point to closest point on bcurve

PK_BCURVE_fit_eval_data_t

```
struct PK_BCURVE_fit_eval_data_s
    {
    PK_POINTER_t            external_data;--- for external evaluator (NULL)

    PK_BCURVE_fit_chain_t   chain_data;   --- for internal chain-of-curves
                                          --- evaluator
    };

typedef struct PK_BCURVE_fit_eval_data_s PK_BCURVE_fit_eval_data_t;
```

This structure contains the data for the fitting evaluator functions.

This structure contains the following fields:

'external_data' This field is to contain data for a user-supplied evaluator function. It will be passed to the function whenever that function is called. Parasolid will not examine this data.

'chain_data' This field is to contain data for the internal 'PK_BCURVE_fit_eval_chain_c' evaluator function.

PK_BCURVE_fit_eval_f_t

```
typedef PK_ERROR_code_t(*PK_BCURVE_fit_eval_f_t)
(
--- received arguements ---

double          parameter,          --- parameter at which to evaluate
PK_LOGICAL_t    calculate_deriv,    --- whether the evaluator should return
                                    --- the 1st derivative
PK_POINTER_t    external_data,      --- external application data

--- returned arguements ---

PK_VECTOR_t     *const point,       --- sample point at 'parameter'
double          *const weight,      --- weight of this point
PK_VECTOR_t     *const deriv        --- 1st derivative at 'parameter'

);
```

This is the type definition for the external evaluator functions expected by PK_BCURVE_create_by_fitting. The functions themselves are to be provided by the application.

Given a 'parameter' a function of type 'PK_BCURVE_fit_eval_f_t' should return a 3-space vector in 'point'.

If 'calculate_deriv' is 'PK_LOGICAL_true' then 'deriv' should be the 1st derivative vector at the point.

If the bcurve is to be rational, 'weight' should contain the weight value of the point. Note that if the bcurve will not be rational the value of the weight is not examined.

The 'external_data' should be supplied to Parasolid via the 'PK_BCURVE_fit_eval_data_t' structure.

Functions of this type should return 'PK_ERROR_no_errors' on success. Any other return value will be taken to be an evaluation failure and the return arguments will not be examined. In these circumstances, a value of

'PK_BCURVE_fit_eval_failure_c' will be returned in the corresponding array entry of the 'status' field of the 'PK_BCURVE_fit_fault_t' 'fault' structure returned by PK_BCURVE_create_by_fitting.

PK_BCURVE_fit_eval_type_t

```
typedef int PK_BCURVE_fit_eval_type_t;
```

This data type selects how sample data for the fitting algorithm will be obtained. It has the values:

```
PK_BCURVE_fit_eval_user_c       use user-supplied evaluator function
PK_BCURVE_fit_eval_chain_c      use internal chain-of-curves evaluator
```

PK_BCURVE_fit_fault_t

```
struct PK_BCURVE_fit_fault_s
    {
    PK_LOGICAL_t     no_faults;          --- true if the fit succeeded
    PK_BCURVE_fit_t *status;             --- status for each bcurve
    double          *distance_errors;    --- final maximum distance errors
    double          *angular_errors;     --- final maximum angular errors
    double          *error_parameters;   --- parameters at which the
                                         --- algorithm failed
    };

typedef struct PK_BCURVE_fit_fault_s PK_BCURVE_fit_fault_t;
```

This structure returns fitting statuses and related data.

This structure has four fields that are arrays. The length of each of these arrays is equal to the 'n_curves' field of the received 'PK_BCURVE_create_by_fitting_o_t' option structure.

The structure's fields are:

'no_faults' this will have the value 'PK_LOGICAL_true' if the fitting completed as requested i.e. all bcurves have been created, are valid and are within the tolerance(s) provided.

'status' an array of length 'n_curves' specifying the final status of the attempt to fit each bcurve.

'distance_errors' an array of length 'n_curves' containing the maximum distance error for each bcurve fitted. In a successful fitting these distances will be less than the 'tolerance' supplied. If fitting failed these values may be undefined, depending on the 'status' for the bcurve.

'angular_errors' an array of length 'n_curves' containing the maximum angular error for each bcurve fitted, if a non-zero angular tolerance was provided. If no angular tolerance was supplied these values will be zero. If fitting failed these values may be undefined, depending on the 'status' for the bcurve.

'error_parameters' an array of length 'n_curves' containing the parameter(s) on the bcurves at which the fitting algorithm has failed, if such a failure has occured. Error parameters will only be returned for certain 'status' values and should be interpreted accordingly.

The 'status' values which return bcurves and associated 'distance_errors' and 'angular_errors' are:

```
'PK_BCURVE_fit_success_c'
'PK_BCURVE_fit_other_failed_c'
'PK_BCURVE_fit_eval_failure_c'
```

The 'status' values which return associated 'error_parameters' and the appropriate interpretations are:

'PK_BCURVE_fit_eval_failure_c'	the parameter at which the evaluator failed
'PK_BCURVE_fit_other_failed_c'	the parameter at which the other bcurve failed
'PK_BCURVE_fit_bad_break_parm_c'	the bad break parameter
'PK_BCURVE_fit_bad_fit_parm_c'	the bad fit parameter

PK_BCURVE_fit_t

typedef int PK_BCURVE_fit_t;

This data type denotes the final status of the attempt to fit a bcurve. It has values:

PK_BCURVE_fit_success_c	fit succeeded
PK_BCURVE_fit_failure_c	unspecified failure
PK_BCURVE_fit_other_failed_c	failed due to another curve failing
PK_BCURVE_fit_eval_failure_c	evaluator function failed
PK_BCURVE_fit_bad_n_curves_c	invalid number of curves supplied
PK_BCURVE_fit_bad_range_c	invalid range supplied
PK_BCURVE_fit_bad_tolerance_c	invalid tolerance supplied
PK_BCURVE_fit_bad_err_method_c	invalid error method supplied
PK_BCURVE_fit_bad_eval_type_c	invalid eval type supplied
PK_BCURVE_fit_bad_break_parm_c	invalid break parameters supplied
PK_BCURVE_fit_bad_chain_c	invalid chain data supplied
PK_BCURVE_fit_bad_fit_parm_c	invalid fit parameters supplied

PK_BCURVE_form_t

typedef int PK_BCURVE_form_t;

This field can take any one of the following values.

```
'PK_BCURVE_form_unset_c'      : The curve shape has not been calculated.
'PK_BCURVE_form_arbitrary_c'  : The curve shape is not special.
'PK_BCURVE_form_polyline_c'   : The curve is piecewise linear.
'PK_BCURVE_form_circular_c'   : The curve corresponds to a circular arc.
'PK_BCURVE_form_elliptic_c'   : The curve corresponds to an elliptic arc.
'PK_BCURVE_form_parabolic_c'  : The curve corresponds to a parabolic arc.
'PK_BCURVE_form_hyperbolic_c' : The curve corresponds to a hyperbolic arc.
```

PK_BCURVE_join_o_t

```
struct PK_BCURVE_join_o_s
    {
    int         o_t_version;        --- version of options structure (1)
    int         n_knots;            --- number of knot values (0)
    double     *knots;              --- acceptable values to snap to (NULL)
    double      snap_tolerance;     --- tolerance for snapping to knots (0.0)
    };

typedef struct PK_BCURVE_join_o_s PK_BCURVE_join_o_t;
```

Holds optional controls on the joining of bcurves.

The option structure defines options applicable to the joining of bcurves.

PK_BCURVE_make_bsurf_lofted_o_t

```
struct PK_BCURVE_make_bsurf_lofted_o_s
    {
    int              o_t_version;     --- version number of option structure(1)
    PK_PARAM_end_t   v_end;           --- end condition at lower v end
                                      --- (PK_PARAM_end_natural_c)
    PK_PARAM_end_t   V_end;           --- end condition at upper V end
                                      --- (PK_PARAM_end_natural_c)
    int              n_v_derivs;      --- number of doubles in v_derivs (0)
    int              n_V_derivs;      --- number of doubles in V_derivs (0)
    double          *v_derivs;        --- derivative data at lower v (NULL)
    double          *V_derivs;        --- derivative data at upper V (NULL)
    PK_PARAM_twist_t u_v;             --- how lower u lower v twist vector
                                      --- defined (PK_PARAM_twist_no_c)
    PK_PARAM_twist_t u_V;             --- how lower u upper V twist vector
                                      --- defined (PK_PARAM_twist_no_c)
    PK_PARAM_twist_t U_v;             --- how upper U lower v twist vector
                                      --- defined (PK_PARAM_twist_no_c)
    PK_PARAM_twist_t U_V;             --- how upper U upper V twist vector
                                      --- defined (PK_PARAM_twist_no_c)
    PK_VECTOR_t      u_v_twist;       --- lower u lower v twist vector (0,0,0)
    PK_VECTOR_t      u_V_twist;       --- lower u upper V twist vector (0,0,0)
    PK_VECTOR_t      U_v_twist;       --- upper U lower v twist vector (0,0,0)
    PK_VECTOR_t      U_V_twist;       --- upper U upper V twist vector (0,0,0)
    PK_PARAM_degen_t v_degen;         --- degeneracy beyond lower v
                                      --- (PK_PARAM_degen_no_c)
    PK_PARAM_degen_t V_degen;         --- degeneracy beyond upper V
                                      --- (PK_PARAM_degen_no_c)
    PK_VECTOR_t      v_degen_vector;  --- position vector of point degeneracy
                                      --- (0,0,0)
    PK_VECTOR_t      V_degen_vector;  --- position vector of point degeneracy
                                      --- (0,0,0)
    int              n_v_degen_bcurve;--- number of doubles in v_degen_bcurve
                                      --- (0)
    int              n_V_degen_bcurve;--- number of doubles in V_degen_bcurve
                                      --- (0)
    double          *v_degen_bcurve;  --- vertices of degenerate bcurve (NULL)
    double          *V_degen_bcurve;  --- vertices of degenerate bcurve (NULL)
    PK_PARAM_knot_t  how_knot;        --- how knot vector defined
                                      --- (PK_PARAM_knot_auto_c)
    double          *knot;            --- knot vector (NULL)
    PK_LOGICAL_t     amalgamate;      --- whether to amalgamate knot vectors
                                      --- (PK_LOGICAL_false)
    int              n_v_degen_segs;  --- number of degenerate segments to add
                                      --- to lower v curve (0)
    int              n_V_degen_segs;  --- number of degenerate segments to add
                                      --- to upper V curve (0)
```

```
    int             *v_degen_segs;    --- degenerate segment positions on lower
                                      --- v curve (NULL)
    int             *V_degen_segs;    --- degenerate segment positions on upper
                                      --- V curve (NULL)
    PK_LOGICAL_t     force_cubic;     --- make result cubic in v even if it is
                                      --- a ruled surface
                                      --- (PK_LOGICAL_false)
    };

typedef struct PK_BCURVE_make_bsurf_lofted_o_s PK_BCURVE_make_bsurf_lofted_o_t;
```

This data structure is the options structure for PK_BCURVE_make_bsurf_lofted.

Specific Errors

PK_ERROR_bad_derivative	derivative or twist vector too large
PK_ERROR_wrong_number_derivs	wrong number of derivatives
PK_ERROR_bad_deriv_vertices	wrong number of vertices of derivative bcurve
PK_ERROR_bad_degen_vertices	wrong number of vertices of degenerate bcurve
PK_ERROR_incompatible_props	only one of 'v_end' and 'V_end' is PK_PARAM_end_periodic_c, 'v_end' is PK_PARAM_end_periodic_c and 'v_degen' or 'V_degen' is not PK_PARAM_degen_no_c, one end is PK_PARAM_end_natural_c and a twist vector has been supplied, 'amalgamate' is PK_LOGICAL_true and 'n_v_degen_segs' or 'n_V_degen_segs' is not 0, an end has both degen_segs and degeneracy beyond the edge.
PK_ERROR_bad_index	index for degenerate segment out of range
PK_ERROR_bad_knots	curve knot vectors are incompatible, bad supplied knot vector

The lofted surface has its lower u edge along the start points of the supplied bcurves and its upper U edge along the end points of the supplied bcurves. It has its lower v edge along the first supplied bcurve and its upper V edge along the last supplied bcurve.

'v_end'	end condition at lower v (ie at first bcurve) edge of surface. If this is set to PK_PARAM_end_periodic_c then 'V_end' must also be set to PK_PARAM_end_periodic_c. If it is set to PK_PARAM_end_clamped_vector_c then derivatives across the lower v edge must be provided in 'v_derivs'. If it is set to PK_PARAM_end_clamped_bcurve_c then the vertices of a derivative bcurve must be supplied in 'v_derivs'.
'V_end'	end condition at upper V (ie at last bcurve) edge of surface. If this is set to PK_PARAM_end_periodic_c then 'v_end' must also be set to PK_PARAM_end_periodic_c. If it is set to PK_PARAM_end_clamped_vector_c then derivatives across the upper V edge must be provided in 'V_derivs'. If it is set to PK_PARAM_end_clamped_bcurve_c then the vertices of a derivative bcurve must be supplied in 'V_derivs'.
'n_v_derivs'	number of doubles in 'v_derivs'
'n_V_derivs'	number of doubles in 'V_derivs'

'v_derivs'	if derivatives are supplied at the lower v edge this either contains a derivative for each spline point (1) of the first bcurve or the vertices of a derivative bcurve.
'V_derivs'	if derivatives are supplied at the upper V edge this either contains a derivative for each spline point (1) of the last bcurve or the vertices of a derivative bcurve.
'u_v'	whether a twist vector for the lower u lower v corner is supplied
'u_V'	whether a twist vector for the lower u upper V corner is supplied
'U_v'	whether a twist vector for the upper U lower v corner is supplied
'U_V'	whether a twist vector for the upper U upper V corner is supplied
'u_v_twist'	lower u lower v twist vector if 'u_v' is PK_PARAM_twist_yes_c
'u_V_twist'	lower u upper V twist vector if 'u_V' is PK_PARAM_twist_yes_c
'U_v_twist'	upper U lower v twist vector if 'U_v' is PK_PARAM_twist_yes_c
'U_V_twist'	upper U upper V twist vector if 'U_V' is PK_PARAM_twist_yes_c
'v_degen'	whether the surface degenerates to a point beyond the lower v bcurve. If this is set to PK_PARAM_degen_vector_c then a degenerate position vector is supplied in 'v_degen_vector'. If this is set to PK_PARAM_degen_bcurve_c then the control vertices of a degenerate bcurve are supplied in v_degen_bcurve.
'V_degen'	whether the surface degenerates to a point beyond the upper V bcurve. If this is set to PK_PARAM_degen_vector_c then a degenerate position vector is supplied in 'V_degen_vector'. If this is set to PK_PARAM_degen_bcurve_c then the control vertices of a degenerate bcurve are supplied in V_degen_bcurve.
'v_degen_vector'	degenerate position vector if 'v_degen' is PK_PARAM_degen_vector_c.
'V_degen_vector'	degenerate position vector if 'V_degen' is PK_PARAM_degen_vector_c.
'n_v_degen_bcurve'	number of doubles in v_degen_bcurve
'n_V_degen_bcurve'	number of doubles in V_degen_bcurve
'v_degen_bcurve'	vertices of degenerate bcurve if 'v_degen' is PK_PARAM_degen_bcurve_c.
'V_degen_bcurve'	vertices of degenerate bcurve if 'v_degen' is PK_PARAM_degen_bcurve_c.
'how_knot'	whether a knot vector is supplied
'knot'	if 'how_knot' is PK_PARAM_knot_defined_c this contains the knot vector. The number of values in the knot vector is equal to the number of lofted curves for a non-periodic lofting or one greater than the number of lofted curves for a periodic lofting.

'amalgamate'	if this is set to PK_LOGICAL_true then the knot vectors of the supplied curves will be amalgamated to produce a surface which is continuous to the same order as the curves. This option may not be used in conjunction with inserted degenerated segements.
'n_v_degen_segs'	number of degenerate segments added to lower v (ie first) bcurve.
'n_V_degen_segs'	number of degenerate segments added to upper V (ie last) bcurve.
'v_degen_segs'	if the first bcurve has fewer segments then the other curves then this may be used to add the required number of degenerate segments. Each integer value given here identifies the segment preceding a required additional degenerate segment. Values must be given in ascending order. Previously inserted degenerate segments affect the segment numbers of subsequent ones.
'V_degen_segs'	as for 'v_degen_segs' for the upper V (ie last) bcurve.
'force_cubic'	in general, the lofted surface will be cubic in the loft (v) direction. However if only two curves are given and both 'v_end' and 'V_end' are PK_PARAM_end_natural_c then a ruled surface which is linear in v will be produced if 'force_cubic' is PK_LOGICAL_false. If 'force_cubic' is set to PK_LOGICAL_true such a surface will be made cubic in the v direction.

(1) a 'spline point' of a bcurve is a point between adjacent segments of of the bcurve or the start or end of the curve.

PK_BCURVE_make_matched_o_t

```
struct PK_BCURVE_make_matched_o_s
    {
    int             o_t_version;
    };

typedef struct PK_BCURVE_make_matched_o_s PK_BCURVE_make_matched_o_t;
```

Holds optional controls on the matching of bcurves.

The option structure defines options applicable to the matching of bcurves.

The structure has no user fields at present:

PK_BCURVE_piecewise_sf_t

```
struct PK_BCURVE_piecewise_sf_s
    {
    int                   degree;          --- The degree = order-1.
    int                   n_segments;      --- The number of curve segments
    int                   dim;             --- Dimension of coefficient vecs
    PK_LOGICAL_t          is_rational;     --- True if the curve is rational.
    PK_piecewise_rep_t    rep;             --- Representation method.
    double               *coeffs;          --- Coefficient vectors
    };
typedef struct PK_BCURVE_piecewise_sf_s PK_BCURVE_piecewise_sf_t;
```

This data structure is the piecewise standard form of of a b-curve.

Specific Errors

```
    PK_ERROR_bad_value             unreadable value
    PK_ERROR_bad_dimension         illegal 'dim'
    PK_ERROR_bad_order             hermite basis and degree not 3
    PK_ERROR_discontinuous_curve   adjacent segments do not meet
```

'degree' This field specifies the degree of the curve. The minimum degree is 1. If the Hermite basis is used ('rep' = PK_piecewise_rep_hermite_c) then the curve must be cubic ('degree' = 3).

'n_segments' The number of segments in the curve. This must be greater than or equal to 1. Adjacent segments must meet.

'dim' This field gives the number of doubles per coefficient vector in the array 'coeffs'. Allowable values are 3 and 4.

'is_rational' This field is set to true if the curve is rational. In this case the 'dim' field must be one greater than the dimensionality of the curve. At present, if:

    'is_rational' == PK_LOGICAL_false then 'dim' must be 3
    'is_rational' == PK_LOGICAL_true then 'dim' must be 4.

'rep' This field defines the representation method. See documentation of PK_piecewise_rep_t.

'coeffs' Contains ('degree'+1)*'n_segments' vectors of dimension 'dim'. If 'is_rational' == PK_LOGICAL_false, then the vectors are 3-D vectors giving the x, y and z components. If 'is_rational' == PK_LOGICAL_true, 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 interpretation of 'coeffs' depends on the representation method chosen; this is determined by the value of the field 'rep'.

For more information about these representations see the chapter on "The Mathematical Form of B_Geometry" in the Functional Description.

PK_BCURVE_sf_t

```
struct PK_BCURVE_sf_s
    {
    int degree;                          --- The degree = order-1.
    int n_vertices;                      --- The number of vertices.
    int vertex_dim;                      --- The dimension of each vertex.
    PK_LOGICAL_t is_rational;            --- True if the curve is rational.
    double *vertex;                      --- The vertices.
    PK_BCURVE_form_t form;               --- The curve shape.
    int n_knots;                         --- The number of knot values.
    int *knot_mult;                      --- The multiplicity of each knot.
    double *knot;                        --- The distinct knot values.
    PK_knot_type_t knot_type;            --- Enum describing the knot set.
    PK_LOGICAL_t is_periodic;            --- True if the curve is periodic.
```

```
    PK_LOGICAL_t is_closed;                --- True if the curve is closed.
    PK_self_intersect_t self_intersecting; --- Yes no or maybe.
    };
typedef struct PK_BCURVE_sf_s PK_BCURVE_sf_t;
```

This data structure is the standard form for a b-curve. See the documentation below for the relationship between this data structure and STEP.

Specific Errors

```
PK_ERROR_bad_value              unreadable value
PK_ERROR_bad_knots              illegal knot multiplicity
                                or knot set not monotonic increasing
PK_ERROR_wrong_number_knots     knots inconsistent with vertices
PK_ERROR_bad_dimension          illegal 'vertex_dim'
PK_ERROR_weight_le_0            one of the weights is zero or negative
```

Converting between PK_BCURVE_sf_t and STEP:

This form is easily convertible to the STEP standard, but there are a number of important differences, designed to make its use simpler and more efficient.

'vertex dim':

This field gives the number of doubles per vertex in the array 'vertex'. Allowable values are 2, 3, 4. A value of 2 implies a parameter space (SP) curve. A value of 3 implies a 3-d polynomial curve or a rational parameter space curve. A value of 4 implies a 3-d rational curve. The STEP "cartesian_point" entity allows 1, 2 or 3.

'is_rational':

This field is set to true if the curve is rational. In this case the vertex_dim field must be one greater than the dimensionality of the curve, and the weights must be included in the 'vertex' array. The STEP standard uses a distinct entity subtype "rational_b_spline_curve".

'vertex':

If the curve is polynomial, dimension 2 or 3, then the values in this field represent the vertices explicitly, in the order [u1, v1, u2, v2, ...], or [x1, y1, z1, x2, y2, ...] respectively. If it is rational, dimension 4, then the cartesian points are multiplied by the weights, so that the values in this field represent [x1w1, y1w1, z1w1, w1, x2w2, y2w2, ...]. The STEP standard uses a separate array of weights, and does not multiply the cartesian points by the weights.

'form':

See the documentation of 'PK_BCURVE_form_t'.
This field corresponds to the field of the same name in the STEP "b_spline_curve" entity, except for the 'PK_BCURVE_form_unset_c' value which does not exist in the STEP standard.

'knot_mult' and 'knot':

The values in 'knot_mult' give the number of times each knot is to be repeated. The minimum multiplicity allowed for any knot is 1.
The maximum multiplicity allowed other than for the first and last knot is 'degree'. The maximum allowed for the first or last knot is 'degree'+1.

The values in 'knot' must be distinct and form a monotone increasing set.

See the documentation below for an explanation of the required total number of knots.

These arrays correspond exactly to the fields "knot_multiplicities" and "knots" in the STEP entity sub-type "b_spline_curve_with_knots".

'knot_type':

See the documentation for 'PK_knot_type_t'.
The STEP standard does not contain a field corresponding to this. The information described is conveyed by the use of entity subtypes, such as "bezier_curve". The values 'PK_knot_bezier_ends_c' and 'PK_knot_non_uniform_c' have no corresponding subtype. A curve of one of these types would be translated to STEP as a "bspline_curve_with_knots".

'is_periodic':

If this field is set to true, the parametrisation of the curve "wraps around". It implies that the curve is closed, with G1 continuity at the seam. This field does not exist in the STEP standard.

See the documentation below for additional requirements for the knot set of a periodic b-curve.

'is_closed':

This field being set to true does not necessarily imply that the parametrisation is periodic. It corresponds to the field of the same name in the STEP entity "b_spline_curve".

'self_intersecting':

See the documentation of PK_self_intersect_t.
This corresponds to the field "self_intersect" in the STEP entity "b_spline_curve", except that the STEP standard does not provide for an unset value.

The number of knots:

The knot set referred to here is the "expanded knot set" obtained by repeating each value in the array 'knot' the number of times given by the corresponding element in 'knot_mult'.

A NURBS curve of degree n defined over k parameters (k-1 parameter intervals) requires a further n parameters at either end of its knot set in order for the necessary k+n-1 b-spline basis functions to be fully defined. These additional parameters are commonly known as the "imaginary knots". Thus the total number of knots p is related to the total number of basis functions and hence the number m of control vertices by the expression p = m+n+1.

Periodic Curves:

A periodic curve must be so constructed as to preserve G1 continuity (of tangent direction) at the join.

In addition, it is recommended that a periodic curve be constructed with C(n-1) continuity at the join (where n is the degree). This would give C2 continuity for a cubic. This most easily achieved by making the "expanded knot set" "wrap around" in the sense that the intervals between the first n+1 knots (n "imaginary" knots and one "real" knot) should be the same as the corresponding intervals between the last n+1 "real" knots, and the intervals between the last n+1 knots (one "real" knot and n "imaginary" knots) should be the same as those between the first n+1 "real" knots.

The "period" of such a curve is the parameter interval separating the first and last "real" knots.

if indices begin at 0:

$$\text{period} = t_{k+n-1} - t_{n}$$

```
for i = 0 to n:
    t -     period = t
     k+n+i-1          n+i

    t + period = t
     i            k+i-1
```

If the knot_type "PK_knot_smooth_seam_c" is set, the "expanded knot set" is generated to achieve the conditions stated above. If 'is_periodic' is set to PK_LOGICAL_false, "PK_knot_smooth_seam_c" is equivalent to "PK_knot_unset_c".

PK_BCURVE_splinewise_sf_t

```
struct PK_BCURVE_splinewise_sf_s
    {
    int                  degree;          --- The degree = order-1.
    int                  n_positions;     --- The number of positions
    PK_VECTOR_t         *positions;       --- The positions
    PK_PARAM_end_t       t_end;           --- End condition at lower t
    PK_PARAM_end_t       T_end;           --- End condition at upper T
    PK_VECTOR_t          t_deriv;         --- Derivative at lower t
    PK_VECTOR_t          T_deriv;         --- Derivative at upper T
    PK_PARAM_knot_t      how_knot;        --- How knot vector defined
    double              *knot;            --- Knot vector
    };
typedef struct PK_BCURVE_splinewise_sf_s PK_BCURVE_splinewise_sf_t;
```

This data structure is the splinewise standard form of a 3D b-curve.

Specific Errors

PK_ERROR_incompatible_props	only one end is periodic or a periodic curve has coincident end points.
PK_ERROR_bad_derivative	't_deriv' or 'T_deriv' too large
PK_ERROR_insufficient_points	not enough spline points
PK_ERROR_coincident_points	repeated spline points
PK_ERROR_bad_position	a spline point is outside size box
PK_ERROR_bad_knots	bad parameterisation
PK_ERROR_repeated_knots	repeated knots
PK_ERROR_wrong_number_knots	wrong number of knots
PK_ERROR_bad_order	degree < 0 or degree > 3 or degree < 2 and supplied clamping vectors are incorrect

'degree' curve degree. When the standard form is input to Parasolid (PK_BCURVE_create_splinewise), 'degree' may only be set to 3 to produce cubic curves or 0 in which case Parasolid will select an appropriate degree.

'n_positions' number of positions.
If 't_end' is PK_PARAM_end_periodic_c then 'n_positions' must be at least 3, otherwise 'n_positions' must be at least 2.

'positions' array of positions.
Consecutive positions must not coincide.
For a closed, non-periodic curve the start and end positions must coincide.
For a periodic curve the start and end positions must not coincide.

```
't_end'          end condition at lower t (ie start of curve)
                 If this is PK_PARAM_end_periodic_c then 'T_end' must also be
                 PK_PARAM_end_periodic_c.
                 If this is PK_PARAM_end_clamped_vector_c then 't_deriv' must
                 be set to the start derivative.

'T_end'          end condition at upper T (ie end of curve)
                 If this is PK_PARAM_end_periodic_c then 't_end' must also be
                 PK_PARAM_end_periodic_c.
                 If this is PK_PARAM_end_clamped_vector_c then 'T_deriv'
                 must be set to the end derivative.

't_deriv'        derivative at start of curve if 't_end' is
                 PK_PARAM_end_clamped_vector_c

'T_deriv'        derivative at end of curve if 'T_end' is
                 PK_PARAM_end_clamped_vector_c

'how_knot'       how the knot vector is defined

'knot'           if 'how_knot' is PK_PARAM_knot_defined_c then 'knot' is the
                 knot vector of length 'n_positions' if the curve is not
                 periodic or 'n_positions' + 1 if the curve is periodic.
```

PK_BCURVE_t

This type represents a B geometry CURVE.

```
A bcurve is a type of curve (PK_CLASS_bcurve is a subclass of PK_CLASS_curve).

The bcurve is defined by data contained in a PK_BCURVE_sf_t.
Piecewise and splinewise representations of bcurves are also available:
See PK_BCURVE_piecewise_sf_t and PK_BCURVE_splinewise_sf_t
```

PK_blend_check_fa_fa_t

typedef int PK_blend_check_fa_fa_t;

```
This data type controls the checking for face-face inconsistencies
during the blending operation. It has two values:

    PK_blend_check_fa_fa_no_c     don't do any face-face inconsistency checks
    PK_blend_check_fa_fa_yes_c    do all of the face-face inconsistency checks
```

PK_blend_check_fa_t

typedef int PK_blend_check_fa_t;

```
This data type controls the checking of whether the new faces created
during the blending operation are valid. It has two values:

    PK_blend_check_fa_no_c     don't do any face checks
    PK_blend_check_fa_yes_c    perform all of the face checks
```

PK_blend_check_su_X_t

typedef int PK_blend_check_su_X_t;

This data type selects which of the new surfaces will be checked for self-intersections during the blending operation. It has three values:

```
PK_blend_check_su_X_no_c          don't check any surfaces
PK_blend_check_su_X_not_bsurf_c   check all surfaces except B-surfaces
PK_blend_check_su_X_yes_c         check all surfaces
```

PK_blend_check_t

```
typedef int PK_blend_check_t;
```

This data type is used to select which type of blend checking is to be used. It has two values:

```
PK_blend_check_vertex_c           check for consistency at vertices
PK_blend_check_full_c             and ensure blend boundaries are legal
```

PK_blend_constraint_t

```
struct PK_blend_constraint_s
    {
    PK_EDGE_array_t       cliff_edges;       --- cliff edges
    PK_EDGE_array_t       tangent_edges;     --- tangent holdline edges
    PK_EDGE_array_t       conic_edges;       --- conic holdline edges
    PK_EDGE_array_t       inv_tangent_edges; --- inverse tangent holdline edges
    PK_EDGE_array_t       inv_conic_edges;   --- inverse conic holdline edges
    PK_PLANE_t            limit_1;           --- first limit plane
                                             --- (PK_ENTITY_null)
    PK_PLANE_t            limit_2;           --- second limit plane
                                             --- (PK_ENTITY_null)
    };
typedef struct PK_blend_constraint_s PK_blend_constraint_t;
```

This structure records a set of blend constraints.

Blend Constraint Options:

'cliff_edges' Blend is constrained by cliff edges.
An array of edge tags is given.
The blend will be constrained to be cliffedge blend in the region of the given edges.

'tangent_edges' Blend is constrained by tangent hold lines.
An array of edge tags is given.
The blend will be constrained to be a tangent hold line blend in the region of the given edges.
For constant and variable radius rolling ball blends the blend need not be constrained by these edges. If the blend is a variable radius rolling ball blend, the left and right ranges of the law curve must be equal.

'conic_edges' Blend is constrained by conic hold lines.
An array of edge tags is given.
The blend will be constrained to be a conic hold line blend in the region of the given edges.
For constant and variable radius rolling ball blends blend need not be constrained by these edges. If the blend is a curvature continuous blend then 'cliff_edges' and 'tangent_edges' must not be set, and the number of edge tags in the array must be at least two.

'inv_tangent_edges'
Blend is constrained by inverted tangent hold lines.

```
                    An array of edge tags is given.
                    The blend will be constrained to be a tangent
                    hold line blend in the region of the given edges.
                    For constant and variable radius rolling ball blends
                    the blend need not be constrained by these edges. If the
                    blend is a variable radius rolling ball blend, the left
                    and right ranges of the law curve must be equal.

 'inv_conic_edges'
                    Blend is constrained by inverted conic hold lines.
                    An array of edge tags is given.
                    The blend will be constrained to be a conic hold line
                    blend in the region of the given edges.
                    For constant and variable radius rolling ball blends
                    blend need not be constrained by these edges.  If the
                    blend is a curvature continuous blend then 'cliff_edges'
                    and 'tangent_edges' must not be set, and the number of edge
                    tags in the array must be at least two.

 'limit_1'
 'limit_2'          Limit Planes.
                    One or two planes may be specified.
                    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 structure contains the topology which will constrain the blend. All of the PK_EDGE_array_t fields will have their lengths set to zero by the options structure macro. These options are turned on by setting the length non-zero.

For further details on the use of these options please refer to the Face-Face Blending Chapter of the Parasolid Functional Description.

PK_blend_cs_t

```
typedef int PK_blend_cs_t;
```

This data type is obsolete and has been superseded by PK_blend_xs_plane_t. It selects a blend cross section plane. It has three values:

```
    PK_blend_cs_rolling_ball_c      rolling ball blend
    PK_blend_cs_disc_c              disc blend
    PK_blend_cs_isoparameter_c      isoparameter blend
```

PK_blend_edge_shape_t

```
struct PK_blend_edge_shape_s
    {
    int                  n_ranges;   --- number of ranges (0)
    double              *ranges_1;   --- first ranges (0.0)
    double              *ranges_2;   --- second ranges (0.0)
    double              *rhos;       --- ranges (0.0)
    PK_VECTOR_t         *positions;  --- positions where ranges apply
    };
typedef struct PK_blend_edge_shape_s PK_blend_edge_shape_t;
```

This structure defines the shape of a variable radius rolling ball blend on an edge.

```
'n_ranges'     number of positions along the edge at which ranges and rhos
               are defined

'ranges_1[i]'  range of blend off left face at 'positions[i]'.
```

'ranges_2[i]' range of blend off right face at 'positions[i]'.

'rhos[i]' rho value at 'positions[i]'.

'positions' positions on edge at which ranges and rhos are defined.

All the ranges must be greater than twice the maximum precision of the edge and its associated vertices.

The rhos must either be all zero, or all between zero and one.

If the two ranges at each position are equal and all rhos are zero, then the blend will have circular cross section. Otherwise the blend will have conical cross section.

PK_blend_edge_update_t

typedef int PK_blend_edge_update_t;

This data type en/dis-ables some changes to the blending algorithms which might cause differences in model updates between versions. It currently has two values:

PK_blend_edge_update_0_c	for updating v9.1 to v12.1 models
PK_blend_edge_update_1_c	for updating v13.0 models

PK_blend_fault_t

typedef int PK_blend_fault_t;

This data type is used to return blending fault types. In some cases a faulty topological entity is also returned.
It has the following values:

	fault_topol PK_CLASS_...	
PK_blend_fault_no_blend_c		edge has no blend
PK_blend_fault_no_fault_c		blend is OK
PK_blend_fault_tangent_c		blend on tangent edge
PK_blend_fault_singularity_c	vertex	blend ends on surface singularity
PK_blend_fault_obsolete_c		unsupported V1 or V2 blend
PK_blend_fault_vertex_c	vertex	configuration of edges at vertex is too complicated
PK_blend_fault_sheet_c		edge is laminar
PK_blend_fault_general_c	vertex	edge has non-manifold vertex
PK_blend_fault_2_edge_c	vertex	blend ends on illegal 2 edge vertex
PK_blend_fault_chamfer_c		edge geometry unsuited to asymmetric chamfer ranges
PK_blend_fault_conic_c		edge geometry unsuited to asymmetric conic ranges
PK_blend_fault_bsurf_c	face	blend requires invalid extension of B-surface
PK_blend_fault_range_c	edge	range inconsistent with adjacent blended edge
PK_blend_fault_type_c	edge	blend type inconsistent with adjacent blended edge
PK_blend_fault_edge_c	edge	adjoining edge not blended
PK_blend_fault_loop_c		blend completely overlaps edge loop
PK_blend_fault_overlap_c		overlapping blends
PK_blend_fault_overlap_edge_c	edge	unblended edge overlapped by blend
PK_blend_fault_bad_end_c	vertex	undetermined problem at end of

```
                                                blend
    PK_blend_fault_face_c            face       range of blend on face too large
    PK_blend_fault_overlap_end_c     vertex     illegal overlap on end boundary
    PK_blend_fault_end_c             vertex     blend has illegal end boundary
    PK_blend_fault_chamfer_intsec_c  vertex     can't intersect chamfers at vertex
    PK_blend_fault_edge_intsec_c     edge       end boundary intersects unblended
                                                edge
    PK_blend_fault_other_edge_c      edge       illegal blend on other edge
                                                prevented full check
    PK_blend_fault_cliffedge_c                  cliff edge blend range
                                                inconsistent with cliff edge
    PK_blend_fault_unknown_c                    unspecified numerical problem
                                                with blend
    PK_blend_fault_self_int_c        face       the blend has produced a
                                                self-intersecting surface
    PK_blend_fault_face_face_c       face       the blend has produced a
                                                face-face inconsistency
```

PK_blend_fix_propagate_t

```
struct PK_blend_fix_propagate_s
    {
    PK_LOGICAL_t     propagate;            --- whether to propagate
                                           --- (PK_LOGICAL_true)
    double           tolerance;            --- propagation angular tolerance
                                           --- (pi/36)
    PK_BODY_blend_propagate_f_t function;  --- user propagation function (NULL)
    PK_POINTER_t     data;                 --- user data for callback function
                                           --- (NULL)
    };
typedef struct PK_blend_fix_propagate_s PK_blend_fix_propagate_t;
```

This structure records information specifying the amount of propagation which shall be performed during a blending operation.

```
This structure contains a collection of switches which will determine
the level of propagation performed during a blending operation.
These control which edges will be propagated to during a blend operation
'propagate'       prevents any propagation if PK_LOGICAL_false.
                  The default value is PK_LOGICAL_true.
'tolerance'       controls the maximum angle (in radians) past which we
                  will propagate. The default value is pi/36.
'function'        is a user callback function called on any edge onto which
                  the operation is going to propagate.
                  The default value is NULL
'data'            is a pointer to the user data for that function.
                  The default value is NULL
If propagation is prevented, either by setting 'propagate' to false or by the
user function returning false for an edge, PK_body_fix_blends will return
PK_blend_fault_edge_c and the tag of the edge onto which it would otherwise
have propagated.
```

PK_blend_group_rib_t

```
typedef int PK_blend_group_rib_t;
```

```
This data type selects how 'max_n_ribs' limits the number of ribs to be
returned.  It has three values:

    PK_blend_group_rib_no_c        don't group ribs
    PK_blend_group_rib_by_face_c   group ribs by blend face
    PK_blend_group_rib_by_parms_c  group ribs by parameter intervals on spine
```

PK_blend_identify_t

```
typedef int PK_blend_identify_t;
```

Terminology:

A "branch face" is any blend face that splits a chain of blends into two or more branches.

A "minimal chain" of blends is terminated by any branch face that is not in the given 'faces'

A "maximal chain" of blends includes any branch faces and the chains emanating from those branch faces.

This datatype allows the user to specify how blend faces are identified from a given set of faces. It has the following values:

PK_blend_identify_within_c	only blend faces within the given face set are identified
PK_blend_identify_exc_chain_c	minimal chains of blends starting within the given face set and excluding final branch faces.
PK_blend_identify_inc_chain_c	minimal chains of blends starting within the given face set and including final branch faces.
PK_blend_identify_max_chain_c	maximal chains of blends starting within the given face set.
PK_blend_identify_dependent_c	blends not in the face set but dependent on faces within the given face set are identified.

More details can be found in choosing how to identify blends

PK_blend_law_t

```
typedef PK_LAW_sf_t PK_blend_law_t;
```

This structure defines a law function over a parameter interval.

See documentation for PK_LAW_sf_t.

PK_blend_local_check_t

```
struct PK_blend_local_check_s
    {
    PK_blend_check_su_X_t  check_su_X;      --- check for self-intersecting
                                            --- surfaces
                                            --- ( PK_blend_check_su_X_no_c )
    PK_blend_check_fa_t    check_fa;        --- check individual faces
                                            --- ( PK_blend_check_fa_no_c )
    PK_blend_check_fa_fa_t check_fa_fa;     --- check for face-face
                                            --- inconsistencies
                                            --- ( PK_blend_check_fa_fa_no_c )
    };
typedef struct PK_blend_local_check_s PK_blend_local_check_t;
```

This structure records information specifying the amount of local-checking which shall be performed during a blending operation.

This structure contains a collection of switches which will determine
the level of extra checking performed during a blending operation.
Switching these checks on can increase the time taken to apply a blend.

'check_surf_X' controls the checking for self-intersecting surfaces.

'check_fa' controls the checking whether individual faces are valid.

'check_fa_fa' controls the checking for face-face inconsistencies.

PK_blend_output_rib_t

typedef int PK_blend_output_rib_t;

This data type selects whether and how blend ribs are output.
It has four values:

PK_blend_output_rib_no_c	don't return ribs
PK_blend_output_rib_on_fail_c	return ribs in failed blend faces
PK_blend_output_rib_only_c	return ribs but no blends
PK_blend_output_rib_also_c	return ribs as well as and blends

PK_blend_ov_cliff_end_t

typedef int PK_blend_ov_cliff_end_t;

This data type is used to indicate how to deal with cliff end edge overflow.
It has two values:

PK_blend_ov_cliff_end_no_c	prevent cliff end edge overflow
PK_blend_ov_cliff_end_yes_c	enable cliff end edge overflow

PK_blend_ov_cliff_t

typedef int PK_blend_ov_cliff_t;

This data type is used to indicate how to deal with cliff edge overflow.
It has three values:

PK_blend_ov_cliff_no_c	prevent cliff edge overflow
PK_blend_ov_cliff_diff_c	enable cliff edge overflow for different convexity
PK_blend_ov_cliff_any_c	enable cliff edge overflow for any convexity

PK_blend_ov_notch_t

typedef int PK_blend_ov_notch_t;

This data type is used to indicate how to deal with notch overflow.
It has two values:

PK_blend_ov_notch_no_c	prevent notch overflow
PK_blend_ov_notch_yes_c	enable notch overflow

PK_blend_ov_smooth_t

```
typedef int PK_blend_ov_smooth_t;
```

This data type is used to indicate how to deal with smooth overflow.
It has three values:

PK_blend_ov_smooth_no_c	prevent smooth overflow
PK_blend_ov_smooth_diff_c	enable smooth overflow for different convexity
PK_blend_ov_smooth_any_c	enable smooth overflow for any convexity

PK_blend_propagate_t

```
typedef int PK_blend_propagate_t;
```

This data type is used to indicate whether the blend is to be propagated.
It has two values:

PK_blend_propagate_no_c	do not propagate
PK_blend_propagate_yes_c	propagate

PK_blend_properties_t

```
struct PK_blend_properties_s
    {
    PK_blend_propagate_t    propagate;     --- whether to progagate
                                           --- (PK_blend_propagate_no_c)
    PK_blend_vary_t         vary;          --- form of variation
                                           --- (PK_blend_vary_smooth_c)
    PK_blend_render_rib_t   render_rib;    --- enable renering of ribs
                                           --- (PK_blend_render_rib_no_c)
    PK_LOGICAL_t            draw_fix;      --- whether blend will be drawn
                                           --- and fixed (PK_LOGICAL_true)
    PK_blend_ov_smooth_t    ov_smooth;     --- enable smooth overflow
                                           --- (PK_blend_ov_smooth_diff_c)
    PK_blend_ov_cliff_t     ov_cliff;      --- enable cliff edge overflow
                                           --- (PK_blend_ov_cliff_diff_c)
    PK_blend_ov_cliff_end_t ov_cliff_end;  --- enable cliff edge end overflow
                                           --- (PK_blend_ov_cliff_end_no_c)
    PK_blend_ov_notch_t     ov_notch;      --- enable notch overflow
                                           --- (PK_blend_ov_notch_yes_c)
    double                  tolerance;     --- tolerance for blended edge
                                           --- (1.0e-5)
    double                  ribspace;      --- approx separation of rib lines
                                           --- (0.0)
    };
typedef struct PK_blend_properties_s PK_blend_properties_t;
```

This structure records a set of blend properties.

PK_blend_render_rib_t

```
typedef int PK_blend_render_rib_t;
```

This data type is used to indicate whether to draw ribs.
It has two values:

```
    PK_blend_render_rib_no_c            don't draw ribs
    PK_blend_render_rib_yes_c           draw ribs - separation 'ribspace'
```

PK_blend_rib_control_t

```
struct PK_blend_rib_control_s
    {
    PK_blend_output_rib_t output_rib;       --- whether and how ribs are made
                                            --- (PK_blend_output_rib_no_c)
    PK_blend_group_rib_t  group_rib;        --- how to group ribs
                                            --- (PK_blend_group_rib_no_c)
    int                   max_n_ribs;       --- limit ribs made per group (0)
    int                   n_interval_parms; --- number of parameters (0)
    double               *interval_parms;   --- parameters dividing the spine
                                            --- into intervals (NULL)
    };
typedef struct PK_blend_rib_control_s PK_blend_rib_control_t;
```

This structure controls how blend ribs are output.

```
                    Blend Rib Controls.

 'output_rib'       Control how ribs are produced.
                    This token controls under what circumstances face-face
                    blending will produce ribs.  The default is to not
                    output ribs.

 'group_rib'        Control how to group the ribs produced.
                    Grouping controls how the limit on the number of ribs
                    is applied.  This token defines whether the ribs are
                    grouped by blend face, or by the intervals along the
                    'parameter' curve defined by 'interval_parms',
                    or not at all (default).

 'max_n_ribs'       Limit on number of ribs produced per group.
                    The number of ribs produced in each group will be no
                    greater than this number. The default value is zero,
                    for which no ribs will be produced.

 'n_interval_parms' Number of 'interval_parms' supplied.
                    Default is zero.

 'interval_parms'   Interval parameters.
                    These are used to divide the 'parameter' curve into
                    intervals for grouping ribs.  They should have values
                    in the parameter range of the curve.

If 'output_rib' is not set to PK_blend_output_rib_no_c then a curve must be
supplied in the 'parameter' field of the 'shape' structure.

If 'group_rib' is set to PK_blend_group_by_parms and 'n_interval_parms' is
zero, or either zero or one for closed blends, then the effect of 'max_n_ribs'
will be identical to having no grouping.  If 'group_rib' is set to another
value the 'interval_parms' will have no effect.

'max_n_ribs' is an upper bound on the number of ribs produced per group, and
not necessarily the number that will be returned.

For further details on the use of these options please refer to the Face-Face
Blending Chapter of the Parasolid Functional Description.
```

PK_blend_rib_r_t

```
struct PK_blend_rib_r_s
    {
    int                 n_ribs;       --- number of ribs
    int                *rib_indices;  --- unique integer ids for each rib
    double             *rib_parms;    --- parameter of each rib
    PK_GEOM_t          *ribs;         --- ribs
    };
typedef struct PK_blend_rib_r_s PK_blend_rib_r_t;
```

This structure returns blend ribs and corresponding parameters.

The PK_blend_rib_r_t structure contains the fields:

'n_ribs'	the number of curves and points in the 'ribs' array (may be 0)
'rib_indices'	an array of integers uniquely identifying each rib
'rib_parms'	a parallel array of parameters for ribs, with values in the range of the 'parameter' curve
'ribs'	an array of curves and points which are cross-sections of the blend surfaces

The 'ribs' will be points in places where a blend is degenerate or of zero radius. The 'rib_parms' are the parameters on the user-supplied 'parameter' curve at which the corresponding cross-section in 'ribs' was made. The 'rib_indices' are unique integers to identify each rib. They are of the form XYYY where YYY is the number of the rib in blend sheet X.

Any space allocated by Parasolid within this structure may be freed by calling PK_blend_rib_r_f.

PK_blend_setback_data_t

```
struct PK_blend_setback_data_s
    {
    int                  n_edges;   --- number of edges (0)
    const PK_EDGE_t    * edges;     --- edge (NULL)
    const PK_LOGICAL_t * which_end; --- whether setback is on
                                    --- vertex at end of edge (NULL)
    const double       * distances; --- setback distance (NULL)
    };
typedef struct PK_blend_setback_data_s PK_blend_setback_data_t;
```

This structure records information specifying whether the end of a blend on an edge will be setback.

This structure contains data which is used to define the shape of the blend where three or more blends meet at a vertex. Specifying a setback will result in the blend being trimmed back before it meets the blends on neighbouring edges, creating a larger patch in the gap left.

'edges'	is an array of length 'n_edges' specifying on which edges the blend shall be setback.
'which_end'	is an array of length 'n_edges' specifying on which end of the corresponding edge the setback shall occur. If 'which_end[i]' is set to 'PK_LOGICAL_true', the setback shall occur at the vertex at the end of the edge. If 'which_end[i]' is set to 'PK_LOGICAL_false', the setback shall occur at the vertex at the start of the edge.
'distances'	is an array of 'n_edges' non-negative doubles specifying the distance in which the blend should be setback on the corresponding edge.

```
                The blend shall be setback by approximately this distance
                measured from the place where the blend meets the blends
                on its two neighouring edges.
```

PK_blend_shape_t

```
struct PK_blend_shape_s
    {
    PK_blend_xs_plane_t   xsection;   --- cross section plane
                                      --- (PK_blend_xs_rolling_ball_c)
    double                radius;     --- constant radius
                                      --- (0.0)
    PK_CURVE_t            parameter;  --- 3D parameter curve
                                      --- (PK_ENTITY_null)
    PK_LAW_sf_t           range1;     --- 1D law function;
                                      --- ranges off left
    PK_LAW_sf_t           range2;     --- 1D law function;
                                      --- ranges off right
    PK_LAW_sf_t           rho;        --- 1D law function;
                                      --- rhos for general conics
    PK_BCURVE_t           depth;      --- 2D law curve; curvature cts blends
                                      --- (PK_ENTITY_null)
    double                width;      --- width of blend for Constant Width
                                      --- blends (0.0)
    double                ratio;      --- ratio of blend ranges for Constant
                                      --- Width blends (1.0)
    PK_blend_xs_shape_t   xs_shape;   --- cross section shape
                                      --- (PK_blend_xs_shape_unset_c)
    };
typedef struct PK_blend_shape_s PK_blend_shape_t;
```

This structure defines the shape of a blend.

```
                        Blend Shape Options:

 'xsection'   Cross section plane, may be one of:
              PK_blend_xs_rolling_ball_c (default) or
              PK_blend_xs_disc_c or
              PK_blend_xs_isoparameter_c

 'radius'     Radius of blend for defining a constant radius blend
              (default 0.0). A value of 0.0 will be taken to mean
              that this field is unset.

 'parameter'  3D parameter curve.

 'range1'     1D law function; ranges off left.  If this is supplied
              a parameter curve must be provided in 'parameter'.

 'range2'     1D law function; ranges off right.  If this is supplied
              a parameter curve must be provided in 'parameter'.

 'rho'        1D law function; rhos for general conics.  If this is supplied
              a parameter curve must be provided in 'parameter'.

 'depth'      2D law curve for curvature continuous blends.  If this is
              supplied a parameter curve must be provided in 'parameter'.

 'width'      Width of blend for defining constant width blends (default 0.0).
              A value of 0.0 will be taken to mean that this field is unset.

 'ratio'      Ratio of blend ranges for constant width blends.  The ratio is
              defined as "ranges off left" / "ranges off right".  The default
              is 1.0 i.e. equal ranges.  This field will be ignored unless the
              'width' field is set.
```

```
 'xs_shape'   Cross section shape, may be one of:
              PK_blend_xs_shape_unset_c (default)
              PK_blend_xs_shape_conic_c
              PK_blend_xs_shape_g2_c
              PK_blend_xs_shape_chamfer_c
```

If no radius, width or range functions are defined then the blend will be assumed to be determined by its constraints, for example tangent-holdline blends.

If the 'width' field is set to a non-zero value, then 'xsection' must be set to 'PK_blend_xs_rolling_ball_c'.

When more than one of 'range1', 'range2' and 'rho' are set, they must have equal degree, periodicity, closure and knot set. The 'depth' option is incompatible with the 'rho' option.

For further details on the use of these options please refer to the Face-Face Blending Chapter of the Parasolid Functional Description.

PK_blend_transfer_topol_t

typedef int PK_blend_transfer_topol_t;

This data type selects whether or not an attempt will be made to transfer topology in the region completely overlapped by the blend into the blend faces.

It has the two values:

```
PK_blend_transfer_topol_no_c  delete all topology in the blended out region
PK_blend_transfer_topol_yes_c preserve topology in the blended out region
```

PK_blend_trim_t

typedef int PK_blend_trim_t;

This data type is used to indicate how to trim the blend
It has four values:

PK_blend_trim_no_c	Do not trim blend. The blend will not be trimmed to the wall extent. Its end boundaries will be constant parameter lines.
PK_blend_trim_short_c	Short trim to walls. The blend end boundaries will be the constant parameter lines determined by the wall boundaries such that the blend is as short as possible.
PK_blend_trim_long_c	Long trim to walls. The blend end boundaries will be the constant parameter lines determined by the wall boundaries such that the blend is as long as possible.
PK_blend_trim_to_walls_c	Trim blend to walls. The blend end boundaries will be determined by the wall boundaries. (default)

PK_blend_type_t

```
typedef int PK_blend_type_t;
```

This data type indicates the type of blend on an edge.
It has four values:

```
PK_blend_type_no_c                not blended
PK_blend_type_ball_constant_c     constant radius rolling ball blend
PK_blend_type_ball_variable_c     variable radius rolling ball blend
PK_blend_type_chamfer_c           chamfer
```

PK_blend_update_t

```
typedef int PK_blend_update_t;
```

This data type en/dis-ables some changes to the blending algorithms which might cause differences in model updates between versions. It currently has two values:

```
PK_blend_update_0_c         for updating v9.1 models
PK_blend_update_1_c         for updating v10.0 models
```

PK_blend_vary_t

```
typedef int PK_blend_vary_t;
```

This data type is used to indicate how the blend varies.
It has two values:

```
PK_blend_vary_smooth_c
PK_blend_vary_linear_c
```

PK_blend_vx_blend_data_t

```
struct PK_blend_vx_blend_data_s
    {
    PK_LOGICAL_t         vertex_blend; --- whether to vertex blend
                                       --- (PK_LOGICAL_false)
    int                  n_vertices;   --- number of vertices (0)
    const PK_VERTEX_t   *vertices;     --- array of vertices to blend (NULL)
    };
typedef struct PK_blend_vx_blend_data_s PK_blend_vx_blend_data_t;
```

Vertex blending allows users to blend smoothly past sharp edges of opposite convexity to the blend. This structure contains information specifying whether or not vertex blending is required, and a list of vertices specifying where to apply it.

The structure contains a single switch controlling whether or not vertex blending is required.

'vertex_blend' when set to PK_LOGICAL_true switches vertex blending on. The default is PK_LOGICAL_false.

'vertices' is an array of length 'n_vertices' specifying on which vertices to apply vertex blends.

If vertex blending is switched on and no vertices are supplied all suitable vertices are blended.

PK_blend_walls_t

typedef int PK_blend_walls_t;

It has the values:

PK_blend_walls_trim_no_c	Do not trim walls. The walls will not be trimmed. The blend will not be attached to the walls but will be created as a separate sheet body.
PK_blend_walls_trim_both_c	Trim both walls. 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.
PK_blend_walls_attach_c	Trim walls and attach blend. 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 'left_sense' and 'right_sense' are not the same, the body containing the right wall will be negated in order to produce a consistent result. (default)
PK_blend_walls_solid_c	Trim walls, attach blend and make solid. As for PK_blend_walls_attach_c, with the addition that if the result of the blending operation encloses a volume then a solid body will be created. Otherwise a sheet body will be returned.

PK_blend_xs_plane_t

typedef int PK_blend_xs_plane_t;

This data type selects a blend cross section plane. It has three values:

PK_blend_xs_rolling_ball_c	rolling ball blend
PK_blend_xs_disc_c	disc blend
PK_blend_xs_isoparameter_c	isoparameter blend

PK_blend_xs_shape_t

typedef int PK_blend_xs_shape_t;

This data type selects a blend cross section shape. It has these values:

PK_blend_xs_shape_unset_c	shape is implied by other fields
PK_blend_xs_shape_conic_c	shape is conic, with tangent continuity
PK_blend_xs_shape_g2_c	shape is curvature continuous
PK_blend_xs_shape_chamfer_c	shape is linear, with position continuity

PK_blend_y_blend_data_t

```
struct PK_blend_y_blend_data_s
    {
    PK_LOGICAL_t         y_blend;      --- whether to y blend
                                       --- (PK_LOGICAL_false)
    int                  n_vertices;   --- number of vertices (0)
    const PK_VERTEX_t   *vertices;     --- array of vertices to blend (NULL)
    };
typedef struct PK_blend_y_blend_data_s PK_blend_y_blend_data_t;
```

This structure contains information specifying whether or not y-shaped vertex blends are required, and a list of vertices specifying where to apply them.

When two blends on adjacent edges of opposite convexity meet at a three, or more, edge vertex and the default shape of the blend is to roll one blend over the other, a y-shaped vertex blend offers an alternative shape of the blend at the vertex.

'y_blend' when set to PK_LOGICAL_true switches y-shaped vertex blending on.
The default is PK_LOGICAL_false.

'vertices' is an array of length 'n_vertices' specifying on which vertices to apply y-shaped vertex blends.

If y-shaped vertex blending is switched on and no vertices are supplied then a y-shaped vertex blend will be applied to all suitable vertices.

See Creating Y-shaped blends chapter in the Functional Description.

PK_BLENDSF_array_t

This structure contains an array of entities of type PK_BLENDSF_t

```
typedef struct PK_BLENDSF_array_s
    {
    PK_BLENDSF_t   *array;   --- array of BLEND SurFaces
    int             length;  --- length of array
    }
    PK_BLENDSF_array_t;
```

It has the fields:

'array': array of entities of type PK_BLENDSF_t

'length': the number of entities in 'array'

PK_BLENDSF_sf_t

```
    struct PK_BLENDSF_sf_s
        {
        PK_GEOM_t     geom_1;       --- first supporting geometry
        PK_GEOM_t     geom_2;       --- second supporting geometry
        double        radii[2];     --- blend surface radii
        PK_CURVE_t    spine;        --- blend spine
        PK_INTERVAL_t spine_extent; --- parameter range of spine
        };
    typedef struct PK_BLENDSF_sf_s PK_BLENDSF_sf_t;
```

This structure is the standard form of a blendsf: a rolling ball blend surface.

The surface is defined by:

- Two supporting geometries, 'geom_1' and 'geom_2', to which the blendsf is tangential.
- Two 'radii' whose common magnitude is the radius of the rolling ball and whose algebraic sign determines on which side of each supporting surface the blendsf lies.
- A 'spine' curve which is the locus of the centre of the rolling ball.
- A 'spine_extent', which is the parameter range of the 'spine' over which the blendsf is defined.

'spine' is the intersection of a surface offset from 'geom_1' by radii[0] and a surface offset from 'geom_2' by radii[1]. When a value in 'radii' is positive then the offset is in the same direction as the surface normal (as returned by PK_SURF_eval_with_normal). When a value in 'radii' is negative, then the offset is in the opposite direction to the surface normal.

A blendsf is parameterised as:

$$R(u,v) = C(u) + rX(u)\cos(va(u)) + rY(u)\sin(va(u))$$

where:

- C is the 'spine' curve
- r is the magnitude of 'radii[0]' and 'radii[1]'
- u has the range 'spine_extent'
- R(u,0) and R(u,1) are the curves where the blendsf meets the supporting geometries.
- X(u) is in the direction R(u,0) - C(u)
- X(u) and Y(u) are unit vectors such that dC(u)/du, X(u) and Y(u) form an orthogonal set
- a(u) is the angle subtended by points R(u,0) and R(u,1) at the spine
- 0 <= v <= 1

PK_BLENDSF_t

This type represents a BLEND SurFace.

A blend surface is a type of surface (PK_CLASS_blendsf is a subclass of PK_CLASS_surf).

The blend surface is defined by data contained in a PK_BLENDSF_sf_t.

PK_BODY_apply_knit_pattern_o_t

```
struct PK_BODY_apply_knit_pattern_o_s
    {
    int                      o_t_version;
    PK_BODY_sewing_type_t    body_type;        --- (PK_BODY_sewing_any_c)
    PK_LOGICAL_t             sort_face_shells; --- (PK_LOGICAL_true)
    };

typedef struct PK_BODY_apply_knit_pattern_o_s PK_BODY_apply_knit_pattern_o_t;
```

Holds optional controls on the application of a knitting pattern to a collection of bodies.

The option structure defines options applicable to the knitting together of sheet bodies by means of a specified pairing of coincident edges.

The structure has two user fields:

```
    body_type:          Desired type of the resulting body. In cases where
                        the sheets are knitted together to form a body with
                        no laminar edges this permits choosing a sheet or solid
                        result. PK_BODY_sewing_general_c is not a permitted
                        value for this field. The default value is
                        PK_BODY_sewing_any_c.

    sort_face_shells:   If set to PK_LOGICAL_true, PK_BODY_knit_bodies will
                        sort the resulting faces into their respective
                        shells. This field can be set to PK_LOGICAL_false
                        when PK_BODY_knit_bodies is used within a partial
                        Boolean. The default value is PK_LOGICAL_true.
```

PK_BODY_array_t

This structure contains an array of entities of type PK_BODY_t

```
typedef struct PK_BODY_array_s
    {
    PK_BODY_t      *array;   --- array of BODYs
    int             length;  --- length of array
    }
    PK_BODY_array_t;
```

It has the fields:

'array': array of entities of type PK_BODY_t

'length': the number of entities in 'array'

PK_BODY_blend_propagate_f_t

```
typedef PK_LOGICAL_t (*PK_BODY_blend_propagate_f_t)
(
--- received arguments ---

PK_EDGE_t       edge,               --- potential propagation edge
PK_POINTER_t    external_data       --- external application data

);
```

This is the type definition for an external evaluator function to be used by PK_BODY_fix_blends in determining whether to propagate blended edges. The function itself is provided by the application.

Given an EDGE a function of type 'PK_BODY_blend_propagate_f_t' should return a logical indicating whether the blend is to propagate onto that edge or not.

PK_BODY_boolean_o_t

```
struct PK_BODY_boolean_o_s
    {
    int                   o_t_version;     --- version number
    PK_boolean_function_t function;        --- which boolean operation
                                           --- (PK_boolean_unite_c)
    const PK_boolean_config_o_t
                          *configuration;
    const PK_boolean_match_o_t
                          *matched_region;
```

```
    PK_LOGICAL_t          merge_imprinted; --- (PK_LOGICAL_false)
    PK_LOGICAL_t          prune_in_solid;  --- (PK_LOGICAL_false)
    PK_LOGICAL_t          prune_in_void;   --- (PK_LOGICAL_false)
    PK_boolean_fence_t    fence;           --- (PK_boolean_fence_none_c)
    PK_LOGICAL_t          allow_disjoint;  --- (PK_LOGICAL_false)
    PK_LOGICAL_t          selective_merge; --- (PK_LOGICAL_false)
    PK_boolean_check_fa_t check_fa;        --- (PK_boolean_check_fa_yes_c)
    double                default_tol;     --- (1.0E-5)
    double                max_tol;         --- (0.0)
    PK_LOGICAL_t          tracking;        --- (PK_LOGICAL_false)
    PK_LOGICAL_t          merge_attributes;--- (PK_LOGICAL_false)
    PK_LOGICAL_t          keep_target_edges;  --- (PK_LOGICAL_false)
    };
typedef struct PK_BODY_boolean_o_s PK_BODY_boolean_o_t;
```

This option structure is supplied to a boolean operation for bodies.

Description of fields:

'function' — Indicates whether a intersect, unite or subtract operation is to be performed, permitted values are

PK_boolean_intersect_c
PK_boolean_subtract_c
PK_boolean_unite_c

'configuration' — Describes the configuration of target and tool bodies.

'matched_region' — Describes matched regions between the target and tool bodies.

'merge_imprinted' — Merge all mergeable imprinted edges created and/or located by the boolean operation. Note that an existing edge can be considered an imprinted edge if it forms part of the intersection between faces from the target and tool.

'prune_in_solid' — Prune solid regions of the result. Any faces, edges or vertices in the result body which are completely surrounded by solid regions will be deleted.

'prune_in_void' — 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).

'fence' — To determine which resultant bodies are returned when performing solid/sheet (trimming) or sheet/solid (punching) boolean operations.
Permitted values are :
- PK_boolean_fence_front_c
- PK_boolean_fence_back_c
- PK_boolean_fence_none_c

'allow_disjoint' — If this option is set to PK_LOGICAL_true, the result of the boolean will be a single body which may contain disjoint manifold components.
This option is ignored when a solid target and sheet tool body are specified for a subtract operation and the 'fence' option is specified with a value of PK_boolean_fence_none_c.

'selective_merge'	If this is set to PK_LOGICAL_true, the merge phase will avoid mergeing imprinted edges that existed and were mergeable before the start of the boolean. This flag has no effect if 'merge_imprinted' is not set to PK_LOGICAL_true.
'check_fa'	The value of this determines if face checking is performed during the operation, permitted values are : PK_boolean_check_fa_no_c PK_boolean_check_fa_yes_c The default is 'PK_boolean_check_fa_yes_c'. If the face checking is switched off, the boolean operation may produce an invalid body without raising an error. However, in some cases, if face checking is switched on the performance of the boolean operation may be reduced. NOTE: This option is inoperative for PK_BODY_boolean.
'default_tol'	The default tolerance to which the boolean operation, if it has to, may approximate entities or assume that entities are coincident. This value should be set to a value that is related to the tolerance to which the application is creating approximations. If this field is set to zero (0.0), it is assumed that the decision to set tolerances on entities is being left to the modeller.
'max_tol'	This value specifies the maximum tolerance which may be applied to any entity involved in the boolean operation. If this field is set to zero (0.0), the default, then it assumed that no upper limit has been set on the tolerance of any entity. If the field is not set to zero (0.0), then it must be set to a value that is greater than the value of the 'default_tol' field.
'tracking'	Not currently in use.
'merge_attributes'	If this is set to PK_LOGICAL_true, then any situation in which two coincident faces are combined to result in a face in a result body, will be treated as a 'merge' event from the point of view of attribute processing and attribute callbacks. If it is set to PK_LOGICAL_false, which is the default, then such a situation will be treated as the deletion of one of the faces. A subtract operation will never result in merge events.
'keep_target_edges'	This option controls which edge survives when a target edge and a tool edge coincide in a boolean. If both faces adjacent to the tool edge survive, then the tool edge will also survive. Otherwise, the result depends on the setting of the option: PK_LOGICAL_true: The target edge survives, unless the target edge is a new edge imprinted during the boolean, and the tool edge is an original edge - in this case, the tool edge survives.

PK_LOGICAL_false (the default):
The older edge (i.e. the one with the smaller tag) survives.

This option only has an effect if general topology is disabled. If it is enabled, then the target edge will always survive, even if it is a newly imprinted edge.

PK_BODY_check_o_t

```
struct PK_BODY_check_o_s
    {
    int                 o_t_version;  --- version number of option structure
    int                 max_faults;   --- limit on fault numbers (10)
    PK_check_geom_t     geom;         --- bad geometry (PK_check_geom_yes_c)
    PK_check_bgeom_t    bgeom;        --- bad B geometry (PK_check_bgeom_yes_c)
    PK_check_top_geo_t  top_geo;      --- topology/geometry inconsistency
                                      --- (PK_check_top_geo_yes_c)
    PK_check_size_box_t size_box;     --- size box violation
                                      --- (PK_check_size_box_yes_c)
    PK_check_fa_X_t     fa_X;         --- face self intersection
                                      --- (PK_check_fa_X_yes_c)
    PK_check_loops_t    loops;        --- loop checks (PK_check_loops_yes_c)
    PK_check_fa_fa_t    fa_fa;        --- face face inconsistency
                                      --- (PK_check_fa_fa_yes_c)
    PK_check_sh_t       sh;           --- negated or inconsistent shells
                                      --- (PK_check_sh_yes_c)
    PK_check_corrupt_t  corrupt;      --- corrupt data or identifiers
                                      --- (PK_check_corrupt_yes_c)
    };

typedef struct PK_BODY_check_o_s PK_BODY_check_o_t;
```

Holds optional controls for checking a body.

The option structure defines options applicable to checking a body.

PK_BODY_create_sheet_planar_o_t

```
struct PK_BODY_create_sheet_planar_o_s
    {
    int                  o_t_version;    --- version number (1)
    const int           *loops;          --- loop numbers (NULL)
    PK_AXIS1_sf_t        plane;          --- plane (XY plane)
    PK_AXIS2_sf_t        basis_set;      --- local coordinate system
                                         --- (world coordinate)
    };
typedef struct PK_BODY_create_sheet_planar_o_s PK_BODY_create_sheet_planar_o_t;
```

This options structure provides variable form input to PK_BODY_create_sheet_planar.

The boundaries of the planar sheet are defined by a given set of 'loops'. If the 'loops' field is NULL (the default) then all the 'vectors' lie on the same loop, this loop must be defined in a clockwise direction (when viewed in the direction of the surface normal) so as to provide the exterior boundary of the sheet. Any body created must have exactly one loop defined clockwise, all other loops being defined anticlockwise.

In the case where the 'loops' field is specified, for each vector in 'vectors', the corresponding entry in 'loops' is the loop index for the loop to which the vector belongs. Thus the number of elements in each array must be equal.

If the 'loops' field of the options structure is NULL (the default), then the given position vectors form a single loop which is the exterior of the sheet body.

'plane' indicates the plane onto which all the position vectors in 'vectors' are projected.

'plane' and 'vectors' are both defined in the local coordinate system.

The 'basis_set' positions and orientates the local coordinate system in the world coordinate system.

PK_BODY_curve_nmnl_t

```
typedef int PK_BODY_curve_nmnl_t;
```

PK_BODY_curve_nmnl_t has two values:
- PK_BODY_curve_nmnl_no_c Do not create nominal geometry on body
- PK_BODY_curve_nmnl_yes_c Allow creation of nominal geometry body

PK_BODY_extrude_o_t

```
struct PK_BODY_extrude_o_s
    {
    int            o_t_version; --- options structure version number
    PK_bound_def_t start_bound; --- starting bound of swept body
                                --- ( PK_bound_distance_c,
                                ---   PK_LOGICAL_true,
                                ---   0.0,
                                ---   PK_ENTITY_null,
                                ---   PK_LOGICAL_true,
                                ---   1
                                --- )
    PK_bound_def_t end_bound;   --- ending bound of swept body
                                --- ( PK_bound_distance_c,
                                ---   PK_LOGICAL_true,
                                ---   1.0,
                                ---   PK_ENTITY_null,
                                ---   PK_LOGICAL_true,
                                ---   1
                                --- )
    };
typedef struct PK_BODY_extrude_o_s PK_BODY_extrude_o_t;
```

This structure provides options for a linear extrusion of a body.

Its fields are:

'start_bound' how the extruded body is to be bound at the start bound.
See documentation for PK_bound_def_t.

'end_bound' how the extruded body is to be bound at the end bound.
See documentation for PK_bound_def_t.

'start_bound' and 'end_bound' must occur in that order along the given 'path' vector.

PK_BODY_fault_t

```
typedef int PK_BODY_fault_t;
```

This type indicates a fault in one of the PK_BODY_create_<body_type>_topology functions. In most cases a 'fault-index' is also returned which is the index in the 'classes' array of the faulty entity.

PK_BODY_fault_t	Meaning	fault_index
PK_BODY_fault_bad_body_c	PK_CLASS_body appears in classes array but not at the first position	none
PK_BODY_fault_unknown_c	unknown fault	none
PK_BODY_fault_unknown_child_c	unknown child	none
PK_BODY_fault_duplicate_child_c	a parent-child relation is repeated	parent
PK_BODY_fault_wrong_child_c	a child is of the wrong class	child
PK_BODY_fault_missing_child_c	a parent has too few children	parent
PK_BODY_fault_extra_child_c	a parent has too many children	parent
PK_BODY_fault_wrong_parent_c	a vertex has edge and loop parents	vertex
PK_BODY_fault_missing_parent_c	a child has too few parents	child
PK_BODY_fault_extra_parent_c	a child has too many parents	child
PK_BODY_fault_bad_wire_c	disconnected wire - in a wire with two loops, the loops do not match	none
PK_BODY_fault_bad_loop_c	the start and end vertices of the edges in a loop, taking sense into account, do not match up	loop
PK_BODY_fault_bad_shells_c	two faces sharing an edge are in different shells	edge
PK_BODY_fault_disjoint_shell_c	the faces in a shell are not all connected	shell
PK_BODY_fault_non_manifold_c	non-manifold vertex	vertex
PK_BODY_fault_no_fault_c	success - valid topological body created	none

Note that faults are checked for in the order given in the above table, and only the first fault found is returned. Thus if a particular fault is returned, it is known that no faults which precede this in the table are present.

If an edge is used twice as a child with the same sense PK_BODY_fault_extra_parent_c is returned. If an edge is not used with both senses, PK_BODY_fault_missing_parent_c is returned.

If a loop has a mixture of edges and vertices as children PK_BODY_fault_wrong_child_c is returned. If a loop has more than one vertex as children PK_BODY_fault_extra_child_c is returned.

If a vertex has edges and loops as parents, PK_BODY_fault_wrong_parent_c is returned. If a vertex has more than loop as parent, PK_BODY_fault_extra_parent_c is returned.

If a loop has as children a ring edge (one with zero or one vertices) and another edge, PK_BODY_fault_bad_loop_c is returned.

PK_BODY_fill_hole_o_t

```
struct PK_BODY_fill_hole_o_s
    {
    int                     o_t_version; --- version number
    PK_fill_hole_method_t   method;      --- method to use to fill hole
                                         --- (PK_fill_hole_trim_to_hole_c)
    PK_BODY_t               fill_sheet;  --- sheet body to patch over hole
                                         --- (PK_ENTITY_null)
    PK_check_fa_fa_t        check_fa_fa; --- check for face-face inconsistencies
                                         --- (PK_check_fa_fa_yes_c)
    };
typedef struct PK_BODY_fill_hole_o_s PK_BODY_fill_hole_o_t;
```

This structure contains optional controls for PK_BODY_fill_hole

The fields are as follows:

'method' Method to use to fill the hole. Permitted values are:
PK_fill_hole_trim_to_hole_c trim 'fill_sheet' to the boundary of the hole
PK_fill_hole_trim_to_sheet_c trim target bodies to the laminar edges of 'fill_sheet'
PK_fill_hole_create_patch_c create a fill sheet to match the boundary of the hole
The default value is PK_fill_hole_trim_to_hole_c.

'fill_sheet' Sheet body to patch over the hole. Not required for method PK_fill_hole_create_patch_c.

'check_fa_fa' Whether to check for face-face inconsistencies. Permitted values are:
PK_check_fa_fa_yes_c perform face-face checking
PK_check_fa_fa_no_c do not perform any face-face checking
The default value is PK_check_fa_fa_yes_c.

PK_BODY_find_facesets_o_t

```
struct PK_BODY_find_facesets_o_s
    {
    int                 o_t_version;       --- version number of option struct
    PK_boolean_region_t selector;          --- faceset selector
                                           --- (PK_boolean_off_c)
    PK_LOGICAL_t        alternate;         --- whether to return alternating
                                           --- facesets (PK_LOGICAL_false)
    int                 n_selecting_topol; --- no of selecting topologies (0)
    PK_TOPOL_t          *selecting_topol;  --- selecting topologies (NULL)
    PK_LOGICAL_t        want_bounds;       --- whether to return bounding
                                           --- edges (PK_LOGICAL_false)
    };

typedef struct PK_BODY_find_facesets_o_s PK_BODY_find_facesets_o_t;
```

This option structure contains optional controls for PK_BODY_find_facesets.

Description of fields:

'selector' Used to specify which facesets are returned when one or more 'selecting_topol' are specified.
Permitted values are:

o PK_boolean_off_c: all facesets will be returned, regardless of whether any 'selecting_topol' are specified. This is the default.

o PK_boolean_include_c: if 'alternate' is PK_LOGICAL_false, only those facesets that include the specified 'selecting_topol' will be returned.

o PK_boolean_exclude_c: if 'alternate' is PK_LOGICAL_false, only those facesets that do not include the specified 'selecting_topol' will be returned.

See the description of 'alternate' for details about how 'selector' works when 'alternate' is PK_LOGICAL_true.

'alternate' If it is set to PK_LOGICAL_true, alternating facesets, will be returned starting with the supplied 'selecting_topol'.
The 'selecting_topol' must contain only one topology.
The default is PK_LOGICAL_false.

The exact facesets that are returned depends on the value of 'selector', as follows:

o If 'selector' is PK_boolean_include_c, then the facesets containing the 'selecting_topol' are returned, the next outermost facesets are not returned, the next outermost facesets are returned, and so on.

o If 'selector' is PK_boolean_exclude_c, then the facesets containing the 'selecting_topol' ARE NOT returned, the next outermost facesets ARE returned, the next outermost facesets are not returned, and so on.

o If 'selector' is PK_boolean_off_c, all facesets are returned, regardless of the value of 'alternate'.

See Finding facesets in the Functional Description for more information.

'n_selecting_topol' Indicates the number of elements in 'selecting_topol'.

'selecting_topol' Contains topologies that will be used to select the returned facesets (see 'alternate' and 'selector' to understand the selection).
If 'selector' is set to PK_boolean_off_c or if 'selecting_topol' is empty, then all found facesets will be returned.
If 'alternate' is set to PK_LOGICAL_true, then the array must contain only one topology.

If a topology in 'selecting_topol' belongs to several facesets, all the facesets will selected with regards to 'selector'.

'want_bounds' If it is set to PK_LOGICAL_false then no bounding edges will be returned.
If it is set to PK_LOGICAL_true, then for each returned faceset, bounding edges will also be returned.

PK_BODY_find_facesets_r_t

```
struct PK_BODY_find_facesets_r_s
    {
    int               n_selected_facesets;      --- number of returned facesets
    PK_FACE_array_t  *selected_facesets;        --- returned facesets
    PK_EDGE_array_t  *bound_edgesets;           --- edgesets bounding every
                                                --- returned faceset
    };
typedef struct PK_BODY_find_facesets_r_s PK_BODY_find_facesets_r_t;
```

This structure contains the results of PK_BODY_find_facesets. The PK memory associated with this structure may be freed by calling PK_BODY_find_facesets_r_f.

Description of fields:

'n_selected_facesets' The number of returned facesets in the body, according to the selection options.

'selected_facesets'	The returned facesets in the body according to the selection options. Each array element represents a complete faceset.
'bound_edgesets'	The sets of edges bounding returned facesets in the body. Each array element represents a complete edgeset bounding the returned faceset of the same rank in 'selected_facesets'.

PK_BODY_find_knit_pattern_o_t

```
struct PK_BODY_find_knit_pattern_o_s
    {
    int                      o_t_version;
    };

typedef struct PK_BODY_find_knit_pattern_o_s PK_BODY_find_knit_pattern_o_t;
```

Holds optional controls on the execution of determining a knitting pattern for a collection of sheet or solid bodies.

The option structure defines options applicable to determining a pairing of coincident edges by which a collection of sheet bodies may be merged into a larger composite sheet.

The structure has no user fields and is present only to enable future development.

PK_BODY_fix_blends_o_t

```
struct PK_BODY_fix_blends_o_s
    {
    int                        o_t_version;    --- version number of option
                                               --- structure
    PK_LOGICAL_t               local_check;    --- whether to perform local
                                               --- checking ( PK_LOGICAL_false )
    PK_LOGICAL_t               vx_twin;        --- return vertex tag for vertex
                                               --- blend ( PK_LOGICAL_false )
    PK_blend_local_check_t     checks;         --- which local checks to perform
    PK_blend_setback_data_t    setback_data;   --- data defining any setbacks
    PK_blend_transfer_topol_t transfer;        --- whether to move topology to
                                               --- blend
                                               ---(PK_blend_transfer_topol_no_c)
    PK_blend_fix_propagate_t   propagate;      --- whether to propagate
                                               --- blend edges
    PK_blend_vx_blend_data_t   vx_blend_data;  --- data defining any vertex
                                               --- blends
    PK_blend_y_blend_data_t    y_blend_data;   --- data defining any y-shaped
                                               --- vertex blends
    PK_LOGICAL_t               preserve_notch; --- retain notch shape
                                               --- (PK_LOGICAL_false)
    PK_blend_edge_update_t     update;         --- update flag for v13.0
                                               --- (PK_blend_edge_update_O_c)
    };

typedef struct PK_BODY_fix_blends_o_s PK_BODY_fix_blends_o_t;
```

Holds optional controls for fixing blends on a body.

The option structure defines options applicable to fixing blends on a body.

PK_BODY_hollow_o_t

```
struct PK_BODY_hollow_o_s
    {
    int                o_t_version;    --- version number of option structure
    int                n_pierce_faces; --- number of faces in 'pierce_faces'
    const PK_FACE_t   *pierce_faces;   --- faces to be pierced (not offset)
    PK_check_fa_fa_t   check_fa_fa;    --- check for face-face inconsistencies
                                       --- (PK_check_fa_fa_yes_c)
    int                n_offset_faces; --- number of elements in 'offset_faces'
                                       --- and 'offset_values' (0)
    const PK_FACE_t   *offset_faces;   --- faces to be offset by a different
                                       --- value than the function value (NULL)
    const double      *offset_values;  --- offset values for corresponding
                                       --- faces in 'offset_faces' array (NULL)
    PK_LOGICAL_t       pierce_tangent; --- process tangent pierce faces before
                                       --- hollowing operation
                                       --- (PK_LOGICAL_false)
    PK_offset_method_t offset_method;  --- offset method
                                       --- (PK_offset_method_accurate_c)
    };
typedef struct PK_BODY_hollow_o_s PK_BODY_hollow_o_t;
```

This structure contains optional controls for PK_BODY_hollow_2

Description of fields:

'n_pierce_faces' — Length of 'pierce_faces' array. The default value is 0.

'pierce_faces' — The array of faces which are not to be offset. 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. The default value is NULL.

'check_fa_fa' — Check for face-face inconsistencies. Permitted values are:

- PK_check_fa_fa_yes_c
- PK_check_fa_fa_no_c

The default value is PK_check_fa_fa_yes_c.

'n_offset_faces' — Length of 'offset_faces' and 'offset_values' arrays. The default value is 0.

'offset_faces' — The array of faces which are to be offset by a value different to that specified via the function arguments. Every face in this array must be matched with a corresponding value in the 'offset_value' array. Faces listed in this array must not be listed within the 'pierce_faces' array. The default value is NULL.

'offset_values' — The array of specific face offset values. Every value in this array is associated with the corresponding face in the 'offset_faces' array. The same rules apply to these offset values as the 'offset' parameter specified in the function argument list. A zero value is allowed: the associated face will be treated as a pierce face. The default value is NULL.

'pierce_tangent' — A "tangent pierce face" is a pierce face which shares at least one smooth edge with a face which is not a pierce face. When this option is set to PK_LOGICAL_true, any tangent pierce face(s) will be processed before offsetting the faces in the body

so that side faces can be created along the smooth boundary edges. Setting this option to PK_LOGICAL_true has no effect on the result if the body has no tangent pierce faces.

'offset_method' The method used for offsetting surfaces in the body. The default value is PK_offset_method_accurate_c.

PK_BODY_identify_details_o_t

```
struct PK_BODY_identify_details_o_s
    {
    int                     o_t_version;    --- version number of option struct
    PK_double_constraint_t  hole_cyl_radius;
    };

typedef struct PK_BODY_identify_details_o_s PK_BODY_identify_details_o_t;
```

This structure provides options for PK_BODY_identify_details

It has the following fields:

'hole_cyl_radius' a double constraint specifying which cylindrical holes will be identified

PK_BODY_identify_facesets_o_t

```
struct PK_BODY_identify_facesets_o_s
    {
    int                     o_t_version;    --- version number of option struct
    PK_boolean_function_t   function;       --- (PK_boolean_unite_c)
    PK_boolean_fence_t      fence;          --- (PK_boolean_fence_none_c)
    PK_LOGICAL_t            check_manifold; --- (PK_LOGICAL_true)
    };

typedef struct PK_BODY_identify_facesets_o_s PK_BODY_identify_facesets_o_t;
```

This option structure is supplied to the identifying facesets operation.

Description of fields:

'function' Indicates whether a unite, intersect or subtract operation is to be simulated, permitted values are

PK_boolean_unite_c
PK_boolean_intersect_c
PK_boolean_subtract_c

'fence' To determine which resultant bodies are returned when simulatinging solid/sheet (trimming) or sheet/solid (punching) boolean operations.
Permitted values are :
PK_boolean_fence_front_c
PK_boolean_fence_back_c
PK_boolean_fence_none_c

'check_manifold' This controls the check for whether the returned facesets contain a non-manifold condition. If its value is PK_LOGICAL_true, then the checks are carried out and PK_ERROR_non_manifold is returned if the result would be non-manifold. If its value is

PK_LOGICAL_false, then no manifold checks are carried out.

PK_BODY_imprint_cus_normal_o_t

```
struct PK_BODY_imprint_cus_normal_o_s
    {
    int                 o_t_version;  --- version number of option structure
    };
typedef struct PK_BODY_imprint_cus_normal_o_s
               PK_BODY_imprint_cus_normal_o_t;
```

The option structure defines options applicable to imprinting a set of curves on a body.

PK_BODY_imprint_cus_vector_o_t

```
struct PK_BODY_imprint_cus_vector_o_s
    {
    int                 o_t_version;  --- version number of option structure
    };
typedef struct PK_BODY_imprint_cus_vector_o_s
               PK_BODY_imprint_cus_vector_o_t;
```

The option structure defines options applicable to imprinting a set of curves on a body.

PK_BODY_imprint_o_t

```
struct PK_BODY_imprint_o_s
    {
    int             o_t_version;         --- version number of option structure
    PK_LOGICAL_t    imprint_tool;        --- (PK_LOGICAL_true)
    PK_LOGICAL_t    imprint_overlapping; --- (PK_LOGICAL_false)
    const PK_boolean_match_o_t
                    *matched_region;     --- (NULL)
    };

typedef struct PK_BODY_imprint_o_s PK_BODY_imprint_o_t;
```

This option structure is supplied to the body imprinting operation.

Description of fields:

'imprint_tool' — Imprint edges and vertices on the tool body. If this flag is set to PK_LOGICAL_false, edges and vertices are imprinted on the target only. The arrays of edges and vertices on the tool, returned in the results structure, will be set to null.

'imprint_overlapping' — Imprint the boundaries of overlapping areas. This is only necessary when the target and tool are sheet bodies. If a face from the target and a face from the tool are on the same surface, the boundaries of the overlapping area will be imprinted. If this flag is not set, faces on a common surface will not imprint on each other.

PK_BODY_knit_pattern_t

```
struct PK_BODY_knit_pattern_s
    {
    int                          n_edges;
    PK_EDGE_t                   *edges;
    PK_EDGE_t                   *matches;
    int                          n_reversals;
    PK_BODY_t                   *reversals;
    };

typedef struct PK_BODY_knit_pattern_s PK_BODY_knit_pattern_t;
```

A structure describing matches between coincident edges on a given set of bodies. The structure also specifies which owning bodies need to be reversed in order to correctly knit the bodies together.

PK_BODY_knit_pattern_t lists pairs of edges that are tolerantly or accurately coincident and which may be simply joined together by topological operations to assemble a larger composite body.

The structure is normally expected to be completed by a call to PK_BODY_find_knit_pattern, though an application can construct one itself. The structure can be passed to PK_BODY_knit_bodies to specify how a collection of sheet bodies can be assembled into a larger composite body.

edges: An ordered list of edges that can be matched wholly will other edges coincident with them.

matches: An ordered list of edges. Each entry is wholly coincident with the corresponding entry in 'edges'.

reversals: A list of bodies owning edges in either of the 'edges' or 'matches' lists that must be reversed in order that an attempt to assemble a composite body by topologically fusing the paired edges may succeed.

PK_BODY_knit_result_t

```
struct PK_BODY_knit_result_s
    {
    int                          n_unknit_edges;
    PK_EDGE_t                   *unknit_edges;
    PK_BODY_knit_status_t         status;
    };

typedef struct PK_BODY_knit_result_s PK_BODY_knit_result_t;
```

The content of this structure describes the level of completness of the application of the knitting pattern.

PK_BODY_knit_result_t holds information on how close the call to PK_BODY_apply_knit_pattern got to completing the application of the supplied knitting pattern.

unknit_edges: An array of all edges from the knitting pattern that were not knitted together

status: Indicates whether the knitting pattern was fully applied in conformance with the options or not.

PK_BODY_knit_status_t

typedef int PK_BODY_knit_status_t;

This type is used to indicate whether a knitting operation was completed in full compliance with the options applied to it.

PK_BODY_knit_status_t	Meaning
PK_BODY_knit_completed_c	Knitting fully applied the knit pattern in conformance with the received options.
PK_BODY_knit_not_completed_c	Knitting did not fully apply the knit pattern and options.

PK_BODY_loft_clamp_type_t

typedef int PK_BODY_loft_clamp_type_t;

This data type indicates the type of clamp to be applied to the tangents across a profile or guide wire.

PK_BODY_loft_clamp_no_c	The tangents across the profile or guide are not constrained.
PK_BODY_loft_clamp_vector_c	The tangents across the profile or guide are constrained at its vertices to be the values specified
PK_BODY_loft_clamp_face_c	The tangents are specified around the profile or across the guide by those of faces bordering on to it.
PK_BODY_loft_clamp_planar_c	The tangents are constrained to be parallel to a single plane

PK_BODY_loft_curvature_t

typedef int PK_BODY_loft_curvature_t;

This data type indicates the type of curvature control to be exercised at the boundary profile of a lofted sheet

It has two values

PK_BODY_loft_natural_c	Apply natural (zero) curvature across the boundary profile.
PK_BODY_loft_unconstrained_c	Apply no constraint on the curvature across the profile.

PK_BODY_loft_deriv_conds_t

```
struct PK_BODY_loft_deriv_conds_s
    {
    PK_BODY_loft_curvature_t          curvature_condition;
    PK_BODY_loft_clamp_type_t         type;
```

```
    PK_BODY_loft_vector_clamp_t       vector_clamp;
    PK_BODY_loft_face_clamp_t         face_clamp;
    PK_BODY_loft_planar_clamp_t       planar_clamp;
    };

typedef struct PK_BODY_loft_deriv_conds_s PK_BODY_loft_deriv_conds_t;
```

This structure holds controls on the behaviour of the derivatives of the lofted surfaces in the loft direction at a profile, or the profile direction at a guide wire (which one is determined by context).

The derivative conditions structure has the following fields.

'curvature_condition'	Specifies the type of constraint to be applied to curvature across the profile or guide. For PK_BODY_loft_natural_c zero curvature is implied and hence the natural loft end condition is applied there. No further clamp conditions are allowed. You cannot use natural curvature on intermediate profiles, periodic lofts or on guide wires. For PK_BODY_loft_unconstrained_c (the default) no end condition is imposed explicitly on the curvature. Subsequent clamp conditions are then examined.
'type'	Indicates the type of clamp conditions to be applied to the first derivative of the loft surface across the profile or guide. The selection of clamp condition indicates which of the following fields actually holds the description of constraint. For PK_BODY_loft_clamp_vector_c the vector_clamp field is examined for the description. For PK_BODY_loft_clamp_face_c the face_clamp field is examined for the description. For PK_BODY_loft_clamp_planar_c the planar_clamp field is examined for the description. For PK_BODY_loft_clamp_no_c (the default) no field is examined for the description. This is the only allowed value when 'curvature_condition' is set to PK_BODY_loft_natural_c.
'vector_clamp'	Holds the first derivative constraint as a collections of derivatives to be matched at each of the vertices of the profile or guide.
'face_clamp'	Specifies a set of faces neighbouring the profile or guide whose derivatives at their boundary, and across the profile or along the guide, must be matched by the loft surface.
'planar_clamp'	Specifies a normal vector that defines a plane for clamping the loft, and a magnitude that controls the size of the clamped derivatives.

PK_BODY_loft_end_conds_t

```
struct PK_BODY_loft_end_conds_s
    {
    PK_PARAM_periodic_t         periodic;
    PK_BODY_loft_deriv_conds_t  start,
                                end;
    };

typedef struct PK_BODY_loft_end_conds_s PK_BODY_loft_end_conds_t;
```

This structure holds controls on the behaviour of the lofted surfaces at the end profiles. These include control of periodicity and the first and second derivatives of the lofted surfaces in the loft direction.

The end conditions structure has the following fields :

'periodic' Set to PK_PARAM_periodic_yes_c if and only if the start profile is also to be the end profile, and the lofted surfaces are to be periodic in the loft direction. Otherwise this must be set to PK_PARAM_periodic_no_c

'start' Gives the constraints on the first and second derivatives in the loft direction across the start profile.

'end' Gives the constraints on the first and second derivatives in the loft direction across the end profile.

PK_BODY_loft_face_clamp_t

```
struct PK_BODY_loft_face_clamp_s
    {
    int                 n_faces;     --- number of clamping faces and clamped
                                     --- edges (0)
    const PK_FACE_t    *faces;       --- array of clamping faces (NULL)
    const PK_EDGE_t    *edges;       --- array of clamped edges (NULL)
    int                 n_magnitudes;--- number of magnitudes for scaling loft
                                     --- derivatives at clamped vertices (0)
    const double       *magnitudes;  --- array of magnitudes for scaling loft
                                     --- derivatives at clamped vertices (NULL)
    int                 n_vertices;  --- number of vertices with scaling (0)
    const PK_VERTEX_t *vertices;     --- array of vertices with scaling (NULL)
    };

typedef struct PK_BODY_loft_face_clamp_s PK_BODY_loft_face_clamp_t;
```

This structure contains two pairs of parallel arrays. The first pair hold the clamping faces and the edges where face clamps are to be applied. The second pair hold scaling factor magnitudes and the vertices where loft derivatives are scaled relative to the default values.

Across all entries of 'edges', the loft derivatives are determined such that the tangent planes of the loft there are matched with those of the corresponding faces. A default value for the derivative magnitude of the loft derivative is internally determined for every clamped profile or guide vertex. Magnitude control is achievable through the use of vertex and magnitude pair. The actual magnitude of the loft derivative at any vertex specified in 'vertices' is the product of the supplied relative magnitude of the vertex specified as a corresponding entry of 'magnitudes' and the default magnitude value there.

Edges of the profile or guide that are not given in 'edges' are not clamped.

The face clamp structure has the following fields :

'n_faces'	The number of faces held in the structure. This must be equal to or less than the number of edges on the profile or guide.
'faces'	An array of faces, one for each clamped edge of the profile or guide. The faces may belong to different bodies. The faces must be ordered to correspond with the entries of 'edges'.
'edges'	An array of edges where loft derivatives are clamped by corresponding entries of 'faces'. Note that for backwards compatibility purposes, 'edges' may be set to NULL in some cases, which has the same effect as making it the collection of all edges of the profile ordered around the body starting at the edge whose start vertex is the profile's start vertex. The start vertex will be understood to be the first vertex on an open profile, and the specified start vertex on a closed profile. If some edges are to be left unclamped, or in the case of a guide wire, 'edges' may not be set to NULL.
'n_magnitudes'	The number of magnitudes for scaling loft derivatives at vertices specified in 'vertices'.
'magnitudes'	An optional array of doubles, one for each vertex entry in 'vertices' or a single scalar "constant" as scaling factor for the loft derivative at the vertex relative to an internally determined default value. Setting this field to NULL has the same effect as having unity for all individual components of the 'magnitudes' array. This field can be set to NULL only if 'vertices' is also set to NULL. When clamping a profile, if there are guide wires specified for the loft, the loft derivative along any guide wire is completely determined. The entries in 'magnitudes' field for those vertices of the profile that have matches on guide wires must be set to unity to ensure the use of the default values. Similarly, when clamping a guide wire, the entries in 'magnitudes' for those vertices of the guide that have matches on profiles must be set to unity. Magnitudes may only be specified for vertices on an edge which has been face clamped.
'n_vertices'	The number of vertices with clamp magnitude control.
'vertices'	An optional array of vertices where the loft derivatives are multiplied by corresponding entries in 'magnitudes'. This field should be set to NULL if 'magnitudes' is NULL. For backwards compatibility purposes, this field may in some cases be set to NULL when 'magnitudes' is not NULL, which has the same effect as making it the collection of all vertices of the profile ordered around the body starting at the start vertex. In the case of a guide wire, or if some edges are to be left unclamped, then 'vertices' should be explicitly set.

PK_BODY_loft_fault_t

typedef int PK_BODY_loft_fault_t;

A status value to be associated with lofted bodies. It indicates the type of problem encountered during the construction of the body (may be none at all).

PK_BODY_loft_ok_c	Lofted body created without problems
PK_BODY_loft_failure_c	Internal algorithm failure
PK_BODY_loft_bad_profile_c	Invalid loft profile
PK_BODY_loft_small_profile_c	Profile is too small to loft
PK_BODY_loft_surf_failure_c	Failed to created lofted surface
PK_BODY_loft_geom_self_int_c	Lofted body has self intersection in at least one of its geometries.
PK_BODY_loft_geom_degen_c	Lofted body has degeneracy in at least one of its geometries.
PK_BODY_loft_topol_self_int_c	Lofted body has an invalid contact between two of its faces, edges, or vertices.
PK_BODY_loft_check_failure_c	Fault detected in generated body
PK_BODY_loft_bad_match_c	Unsuitable profile mappings detected
PK_BODY_loft_bad_clamp_c	Unsuitable clamps in loft end conditions
PK_BODY_loft_bad_profile_type_c	Profile with incorrect body properties such as type, closure and orientation.
PK_BODY_loft_bad_end_conds_c	General fault detected in end conditions
PK_BODY_loft_bad_guide_wire_c	Fault detected in a guide wire

PK_BODY_loft_matches_t

```
struct PK_BODY_loft_matches_s
    {
    int                          n_mappings;
    PK_BODY_profile_match_t *mappings;
    };

typedef struct PK_BODY_loft_matches_s PK_BODY_loft_matches_t;
```

This structure holds an array of mappings, each of which describes associations between the vertices of a profile of the loft and those of its immediate following neighbour.

The matching structure has the following fields :

'n_mappings'	The number of mappings held in the structure. When mappings are provided his must equal to the number of profiles for a loft that is periodic, and one less than that for a loft that is not periodic. A value of zero for this field indicates no mappings are provided.
'mappings'	An array of mappings, each of which describes associations between the vertices of a profile of the loft and those of its immediately following neighbour.

PK_BODY_loft_one_match_t

```
struct PK_BODY_loft_one_match_s
    {
    PK_VERTEX_t profile_vertex;
    PK_VERTEX_t matched_vertex;
    };

typedef struct PK_BODY_loft_one_match_s PK_BODY_loft_one_match_t;
```

This structure holds an association between a vertex of a given profile of the loft and one on the immediately following profile.

The matching structure has the following fields :

'profile_vertex' A vertex on the profile for which this represents part of the mapping.

'matched_vertex' The corresponding vertex on the following profile.

If there are guide wires for the loft and the geometry of 'profile_vertex' is coincident with a vertex geometry of a guide wire, then the geometry of 'matched_vertex' is required to be also coincident with a vertex geometry of the same guide wire. The coincidences are measured subject to relevant vertex tolerances.

PK_BODY_loft_planar_clamp_t

```
struct PK_BODY_loft_planar_clamp_s
    {
    PK_VECTOR1_t    normal;
    double          magnitude;
    };

typedef struct PK_BODY_loft_planar_clamp_s PK_BODY_loft_planar_clamp_t;
```

This structure holds the normal vector of plane and magnitude of derivatives relative to a default value.

The planar clamp structure has the following fields :

'normal' The normal vector of the plane for clamping the loft. All cross-boundary tangents of the loft at the clamped profile or guide are hence perpendicular to this normal vector. Negating the normal will have the effect of reversing all cross-boundary tangents.

'magnitude' A scalar representing the magnitude of the loft derivative at the clamped profile or guide relative to a default value. If magnitude is set to zero, the default value will be used automatically.

When clamping profiles, and there is a guide wire specified for the loft, the loft derivative along that guide wire is completely determined. The 'magnitude' field must then be set to either unity or zero to ensure the use of the default value there.

When clamping guide wires, the 'magnitude' field must always be set to either unity or zero, since there will always be profiles where the loft derivative is completely determined.

PK_BODY_loft_status_r_t

```
struct PK_BODY_loft_status_r_s
    {
    PK_BODY_loft_fault_t         fault;
    PK_VECTOR_t                  fault_location;
    PK_TOPOL_t                   fault_topol;
    };

typedef struct PK_BODY_loft_status_r_s PK_BODY_loft_status_r_t;
```

The structure packages status and fault location information

The structure has the following fields :

'fault' — A token indicating the degree of success in completing the lofted body. Note that this may indicate failure even where PK_BODY_make_lofted_body returns PK_ERROR_no_errors.

'fault_location' — The three space point where the fault is located, or from where it originated.

'fault_topol' — The topology on which the fault is located, or from which it originated.

PK_BODY_loft_vector_clamp_t

```
struct PK_BODY_loft_vector_clamp_s
    {
    int                 n_tangents;  --- number of derivative vectors (0)
    const PK_VECTOR_t *tangents;     --- array of derivative vectors (NULL)
    int                 n_vertices;  --- number of vertices to be clamped (0)
    const PK_VERTEX_t *vertices;     --- array of vertices to be clamped (NULL)
    };

typedef struct PK_BODY_loft_vector_clamp_s PK_BODY_loft_vector_clamp_t;
```

This structure holds an array of derivative vectors and an optional array of corresponding vertices where the derivative vectors are to be applied.

The vector clamp structure has the following fields :

'n_tangents' — The number of derivatives held in the structure. This must equal the value of 'n_vertices', or be 1.

'tangents' — An array of derivatives one for each vertex of 'vertices', or a single vector "constant". If 'vertices' is set to NULL, they are then taken in order to correspond with the vertices ordered around the profile commencing with the start vertex. Note that 'vertices' may not be set to NULL in the case of a guide wire.

The start vertex will be understood to be the first vertex on an open profile, and the specified start vertex on a closed profile.

'n_vertices' — The number of vertices where the loft derivative is given.

'vertices' — An optional array of vertices where the loft derivative is specified as the corresponding entry in 'tangents'. Setting 'vertices' to NULL has the same effect as setting it to be the collection of all vertices of the profile.

When specifying a vector clamp on a profile, if there are guide wires specified for the loft, the loft derivative along any guide wire is completely determined. Thus, those vertices of the profile that have matches on guide wires must not be included as entries of 'vertices'. Due to the effect of setting it to NULL, 'vertices' must be explicitly assigned to exclude profile vertices that have matches on guide wires.

Similarly, when specifying a vector clamp on a guide wire, the derivative along any profile is completely determined. Thus, those vertices of the guide that have matches on profiles must not be included as entries of 'vertices', and 'vertices' may not be set to NULL.

PK_BODY_make_curves_outline_o_t

```
struct PK_BODY_make_curves_outline_o_s
    {
    int                   o_t_version;     --- version number
    PK_outline_project_t  project;         --- whether to project the curves
                                           --- (PK_outline_project_no_c)
    PK_VECTOR_t           project_position;--- position of plane if projecting
                                           --- (0,0,0)
    double                tolerance;       --- curve tolerance (0.00001)
    PK_LOGICAL_t          want_topols;     --- (PK_LOGICAL_t)
    };
typedef struct PK_BODY_make_curves_outline_o_s PK_BODY_make_curves_outline_o_t;
```

Controls for the PK_BODY_make_curves_outline options structure

The option 'projection' controls whether the outline curves are output as 3-dimensional curves i.e. their depth information is retained, or if they are projected onto a 2-dimensional plane perpendicular to the viewing direction. The position of the plane in the view direction is controlled by 'project_position'. If 'project' is set to PK_outline_project_no_c 'projection_position' has no effect. The option 'tolerance' controls the accuracy of the silhouettes generated. The edges (or faces in the case of silhouettes) corresponding to the curves may also be returned. The option 'want_topols' controls whether the edge or face information is output.

PK_BODY_make_lofted_body_o_t

```
struct PK_BODY_make_lofted_body_o_s
    {
    int                                 o_t_version;
                                                --- version number
    PK_BODY_loft_end_conds_t            end_conditions;
                                                --- behaviour of the lofted
                                                --- surfaces at the end profiles

    int                                 n_guide_wires;
                                                --- number of guide wires (0)

    const PK_BODY_t                    *guide_wires;
                                                --- guide wires (NULL)

    PK_BODY_simplify_t                  simplify;
                                                --- level of simplification
                                                --- (PK_BODY_simplify_yes_c)
    PK_BODY_topology_t                  topology_form;
                                                --- topology of resultant body
```

```
                                        --- (PK_BODY_topology_minimal_c)
    PK_BODY_loft_matches_t            matches;
                                        --- how to match vertices on
                                        --- different profiles

    int                               n_intermediate_derivs;
                                        --- the number of intermediate
                                        --- profiles that have derivative
                                        --- constraints specified (0)
    const PK_BODY_loft_deriv_conds_t *intermediate_derivs;
                                        --- array of derivative
                                        --- constraints (NULL)
    const int                        *intermediate_profiles;
                                        --- array of profile indices
                                        --- (NULL)

    int                               n_guide_derivs;
                                        --- the number of guide wires
                                        --- that have derivative
                                        --- constraints specified (0)
    const PK_BODY_loft_deriv_conds_t *guide_derivs;
                                        --- array of derivative
                                        --- constraints for guide wires
                                        --- (NULL)
    const int                        *guide_indices;
                                        --- array of guide indices
                                        --- (NULL)
    };

typedef struct PK_BODY_make_lofted_body_o_s PK_BODY_make_lofted_body_o_t;
```

This structure holds optional controls on the construction of a lofted body.

The option structure for making a lofted body has the following fields :

'end_conditions'	A structure to specify particular constraints on the derivatives across the start and/or end profiles. It is also used to specify a periodic loft.
'n_guide_wires'	The number of guide curves for the loft.
'guide_wires'	An array of oriented connected manifold wire bodies that specify the guide curves for the loft. Each guide wire is required to cut across all profiles though a set of matched profile vertices at the wire's own vertices, subject to relevant vertex tolerances.
'simplify'	A token used to indicate the level of simplification to be attempted on the lofted surfaces generated to form the faces of the lofted body.
'topology_form'	A token used to indicate how the resultant body should be divided into faces.
'matches'	A structure containing the profile vertex matching information. The structure provides a map between the vertices on one profile with those on the next.
'n_intermediate_derivs'	The number of intermediate profiles (ie, profiles other than the start or end profile) that have derivative conditions specified.
'intermediate_derivs'	An array of length equal to 'n_intermediate_derivs' specifying derivative conditions for each profile as given in 'intermediate_profiles'.
'intermediate_profiles'	An array of length equal to 'n_intermediate_derivs' giving the index of the profile that has the derivative

conditions given by the corresponding entry in 'intermediate_derivs'. Each entry should be an int between 1 and 'n_profiles'-2 for non-periodic lofting, or between 1 and 'n_profiles'-1 for periodic lofting.

'n_guide_derivs' The number of guide wires with derivative constraints. This must be a number between 0 and 'n_guides'.

'guide_derivs' An array of length equal to 'n_guide_derivs' specifying derivative conditions for each guide wire as given in 'guide_indices'.

'guide_indices' An array of length equal to 'n_guide_derivs' giving the index of the guide wire (as given in 'guide_wires') that has the derivative conditions given by the corresponsing entry in 'guide_derivs'.

PK_BODY_make_swept_body_o_t

```
struct PK_BODY_make_swept_body_o_s
    {
    int                        o_t_version;
    PK_BODY_sweep_alignment_t  alignment;     --- alignment of profile
                                              --- (PK_BODY_sweep_align_normal_c)
    PK_BODY_simplify_t         simplify;      --- simplify lofted surfaces
                                              --- (PK_BODY_simplify_yes_c)
    PK_BODY_topology_t         topology_form; --- topology of resultant body
                                              --- (PK_BODY_topology_minimal_c)
    PK_BODY_sweep_law_t        twist;         --- twist of sweep path
    PK_BODY_sweep_law_t        scale;         --- scale of sweep path
    PK_BODY_sweep_fair_t       fair;          --- method to use for improving
                                              --- the sweep
                                              --- (PK_BODY_sweep_fair_no_c)
    };

typedef struct PK_BODY_make_swept_body_o_s PK_BODY_make_swept_body_o_t;
```

This structure holds optional controls on the construction of a swept body.

The option structure for making a swept body has the following fields :

'alignment' Whether the profile normal should maintain a constant angle to the global axes (PK_BODY_sweep_align_parallel_c) or a constant angle to the path tangent (PK_BODY_sweep_align_normal_c)

'simplify' A token used to indicate the level of simplification to be attempted on the lofted surfaces generated to form the faces of the swept body. (PK_BODY_simplify_yes_c)

'topology_form' A token used to indicate how the resultant body should be divided into faces. (PK_BODY_topology_minimal_c)

'twist' A 1-dimensional LAW standard form or a discrete set of values to be interpolated at vertices along the path. The values specify the amount of rotation about the path tangent direction undergone by the profile as it moves along the sweep path.

'scale' A 1-dimensional LAW standard form or a discrete set of values to be interpolated at vertices along the path. The values specify the amount of magnification, centred on the path, of the profile as it moves along the sweep path.

'fair' Indicates what method (if any) to use to attempt to improve the method of sweeping along the supplied path. (PK_BODY_sweep_fair_no_c)

PK_BODY_make_swept_profiles_o_t

```
struct PK_BODY_make_swept_profiles_o_s
    {
    int                          o_t_version;
    PK_BODY_sweep_alignment_t    alignment;
    PK_BODY_simplify_t           simplify;

    PK_BODY_sweep_law_t          twist;
    PK_BODY_sweep_law_t          scale;
    };

typedef struct PK_BODY_make_swept_profiles_o_s PK_BODY_make_swept_profiles_o_t;
```

This structure holds optional controls on the construction of the profiles of a swept body.

The option structure for making the profiles of a swept body has the following fields :

'alignment' Whether the profile normal should maintain a constant angle to the global axes (PK_BODY_sweep_align_parallel_c) or a constant angle to the path tangent (PK_BODY_sweep_align_normal_c)

'simplify' A token used to indicate the level of simplification to be attempted on the lofted surfaces generated to form the faces of the swept body.

'twist' A 1-dimensional LAW standard form or a discrete set of values to be interpolated at vertices along the path. The values specify the amount of rotation about the path tangent direction undergone by the profile as it moves along the sweep path.

'scale' A 1-dimensional LAW standard form or a discrete set of values to be interpolated at vertices along the path. The values specify the amount of magnification, centred on the path, of the profile as it moves along the sweep path.

PK_BODY_offset_o_t

```
struct PK_BODY_offset_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        allow_disjoint; --- whether disjoint body can be
                                        --- produced (PK_LOGICAL_false)
    PK_check_fa_fa_t    check_fa_fa;    --- check for face-face inconsistencies
                                        --- (PK_check_fa_fa_yes_c)
    int                 n_offset_faces; --- number of elements in 'offset_faces'
                                        --- and 'offset_values' (0)
    const PK_FACE_t    *offset_faces;   --- faces to be offset by a different
                                        --- value than the function value (NULL)
    const double       *offset_values;  --- offset values for corresponding
                                        --- faces in 'offset_faces' array (NULL)
    PK_offset_method_t offset_method;   --- offset method
                                        --- (PK_offset_method_accurate_c)
```

```
    PK_offset_step_t   offset_step;    --- whether to create a step offset
                                       --- (PK_offset_step_no_c)
    };
typedef struct PK_BODY_offset_o_s PK_BODY_offset_o_t;
```

This structure contains optional controls for PK_BODY_offset_2

Description of fields:

'allow_disjoint' — If this flag is set to PK_LOGICAL_true, and if the resulting body is split by the operation, all the disjoint components of that body will be preserved within a single body. If this flag is set to PK_LOGICAL_false, only one disjoint part of the body will be returned.
The default value is PK_LOGICAL_false.

'check_fa_fa' — Check for face-face inconsistencies. Permitted values are:
PK_check_fa_fa_yes_c
PK_check_fa_fa_no_c
The default value is PK_check_fa_fa_yes_c.

'n_offset_faces' — Length of 'offset_faces' and 'offset_values' arrays.
The default value is 0.

'offset_faces' — The array of faces which are to be offset by a value different to that specified via the function arguments. Every face in this array must be matched with a corresponding value in the 'offset_value' array.
The default value is NULL.

'offset_values' — The array of specific face offset values. Every value in this array is associated with the corresponding face in the 'offset_faces' array.
The same rules apply to these offset values as the 'offset' parameter specified in the function argument list. A zero value is allowed.
The default value is NULL.

'offset_method' — The method used for offsetting surfaces in the body.
The default value is PK_offset_method_accurate_c.

'offset_step' — A "boundary edge" is an edge between a face which is being offset and one which is not.
If this option is set to PK_offset_step_yes_c then side faces will be created along any smooth boundary edges. Setting this option to PK_offset_step_yes_c has no effect on the result if there are no smooth boundary edges.
The default value is PK_offset_step_no_c.

PK_BODY_offset_planar_wire_o_t

```
struct PK_BODY_offset_planar_wire_o_s
    {
    int             o_t_version;    --- version number
    PK_BODY_owb_gap_fill_t          --- how gaps at vertices are filled
                    gap_fill;       --- (PK_BODY_owb_gap_fill_round_c)
```

```
    PK_LOGICAL_t    local_check;    --- whether to perform local checks
                                    --- (PK_LOGICAL_false)
    double          tolerance;      --- tolerance for curve conversion (0)
    };

typedef struct PK_BODY_offset_planar_wire_o_s PK_BODY_offset_planar_wire_o_t;
```

Options structure for offsetting planar wires.

The fields are as follows:

'gap_fill' specifies how gaps at vertices produced by the offset operation should be filled. Permitted values are:
PK_BODY_owb_gap_fill_round_c fill with round arc (default)
PK_BODY_owb_gap_fill_linear_c fill with tangent extension
PK_BODY_owb_gap_fill_natural_c fill with curve extension

'local_check' determines whether or not to perform local checking for validating the result. The default is not to.

'tolerance' tolerance for converting general curves to bcurves. If this field is set to zero, Parasolid will select a suitable tolerance.

PK_BODY_owb_gap_fill_t

```
typedef int PK_BODY_owb_gap_fill_t;
```

This data type is used to specify how gaps at offset vertices should be filled. It has three values:

```
    PK_BODY_owb_gap_fill_round_c    fill with circular arc
    PK_BODY_owb_gap_fill_linear_c   fill with tangent extension
    PK_BODY_owb_gap_fill_natural_c  fill with natural curve extension
```

PK_BODY_pick_method_t

```
typedef int PK_BODY_pick_method_t;
```

This field can take any one of the following values.

```
    'PK_BODY_pick_radial_c'     : measure pick distance perpendicular to ray
    'PK_BODY_pick_axial_c'      : measure pick distance along ray
    'PK_BODY_pick_ratio_c'      : measure pick distance using blend of both
```

PK_BODY_pick_topol_r_t

```
struct PK_BODY_pick_topol_r_s
       {
       PK_ENTITY_t  entity;
       int          occurence;
       PK_VECTOR_t  intersect;
       double       distance;
       };
typedef struct PK_BODY_pick_topol_r_s
               PK_BODY_pick_topol_r_t;
```

PK_BODY_pick_topols_o_t

```
struct PK_BODY_pick_topols_o_s
    {
--- option structure version number
    int                        o_t_version; --- version number

--- maximum of each type of topological entity to be picked (zero implies none)
    int                           max_faces; --- max number of faces required
                                             --- (1)
    int                           max_edges; --- max number of edges required
                                             --- (0)
    int                        max_vertices; --- max number of vertices required
                                             --- (0)

--- near/far culling
    PK_LOGICAL_t                 near_plane; --- whether set
                                             --- (PK_LOGICAL_false)
    PK_LOGICAL_t                  far_plane; --- whether set
                                             --- (PK_LOGICAL_false)

--- tolerances for the approximation of curves by a sequence of straight lines
    PK_LOGICAL_t        is_curve_chord_tol;
                                             --- whether curve_chord_tol is set
                                             --- (PK_LOGICAL_false)
    double                 curve_chord_tol;
                                             --- tolerance between curve/chord
                                             --- (0.0)
    PK_LOGICAL_t        is_curve_chord_max;
                                             --- whether curve_chord_max is set
                                             --- (PK_LOGICAL_false)
    double                 curve_chord_max;
                                             --- maximum length of a chord
                                             --- (0.0)
    PK_LOGICAL_t        is_curve_chord_ang;
                                             --- whether curve_chord_ang is set
                                             --- (PK_LOGICAL_false)
    double                 curve_chord_ang;
                                             --- max ang between adjacent chords
```

```
                                        --- (0.0)
--- whether to ignore back facing faces (required if max_faces > 0)
    PK_LOGICAL_t         ignore_back_faces; --- face picking optimisation
                                        --- (PK_LOGICAL_true)

--- max distance from edge to ray (required if max_edges > 0)
    double                  max_edge_dist; --- max distance between ray/edges
                                        --- (0.0)

--- max distance from vertex to ray (required if max_vertices > 0)
    double                max_vertex_dist; --- max dist between ray/vertices
                                        --- (0.0)

--- points on near/far planes (required if using near_plane/far_plane options)
    PK_VECTOR_t                near_point; --- point on near_plane
                                        --- (0.0, 0.0, 0.0)
    PK_VECTOR_t                 far_point; --- point on far_plane
                                        --- (0.0, 0.0, 0.0)

--- ordering method
    PK_BODY_pick_method_t           method; --- method of measuring pick dist
                                        --- (PK_BODY_pick_axial_c)

--- ratio of radial:axial distance (required if method = PK_BODY_pick_ratio_c)
    double                           ratio; --- bias when measuring pick dist
                                        --- (0.0)

    int                    n_geom_classes; --- specify the number of
                                        --- geometry classes
                                        --- (0)

    PK_CLASS_t              *geom_classes; --- specify the geometry classes
                                        --- that can be picked
                                        --- NULL

    PK_pick_approximate_t      pick_approx; --- approximate curves
                                        --- or use accurate geometry
                                        --- (PK_pick_approximate_yes_c)
    };
typedef struct PK_BODY_pick_topols_o_s PK_BODY_pick_topols_o_t;
```

Controls picking of topological entities

The PK_BODY_pick_topols_o_t structure provides the following controls :

- the maximum number of each type of entity which can be picked
- the maximum separation allowed between the ray and candidate edges/vertices
- whether picking is performed using a bounded or an unbounded ray
- which chordal tolerances are used to approximate edge curves by polylines
- how distance is measured from candidate entities to the picking ray
- the option to specify which geometry classes can be picked

overview
The max_faces, max_edges and max_vertices options control which types of entity can be picked and the maximum number of each type of entity which can be returned.

Entities can be picked separately or together by setting one or more of the max_faces, max_edges and max_vertices arguments to non-zero values. The default is set by the PK_BODY_pick_topols_o_m macro with max_faces=1 and both max_edges and max_vertices=0 (i.e. set to pick the closest face).

max_faces
If max_faces > 0, this specifies the maximum number of faces which can be picked. To be considered for picking, the intersection point between a candidate face and the ray must lie within (or on) the face boundary.

If a face is intersected by the ray more than once, each intersection is processed separately. Face picking is subject to the 'ignore_back_faces' option.

max_edges
If max_edges > 0, this specifies the maximum number of edges which can be picked. To be picked, the minimum distance between an edge and the ray must be less than or equal to 'max_edge_dist' (this value must be specified).

The function calculates the minimum distance between the edge and the ray (the 'hit distance') and records the position on the ray (the 'hit point') at which this measurement was made.

If other points on the edge lie at the same distance from the ray, only the first case is noted (an edge occurrence can only be picked once).

max_vertices
If max_vertices > 0, this specifies the maximum number of vertices which can be picked. To be picked, the minimum distance between a candidate vertex and the ray must be less than or equal to 'max_vertex_dist' (this value must be specified).

The function calculates the minimum distance between the vertex and the ray (the 'hit distance') and records the position on the ray (the 'hit point') at which this measurement was made.

ignore_back_faces
The 'ignore_back_faces' setting is a performance option which is used when picking faces (max_faces > 0). It enables Parasolid to optimise its search by ignoring back-facing faces of solid bodies. Parasolid will only ignore faces which are wholly facing backwards and it will not necessarily detect all of such cases. This option has no effect when picking faces from sheet bodies.

The PK_BODY_pick_topols_o_m macro sets the 'ignore_back_faces' option to 'PK_LOGICAL_true'. This setting is appropriate when picking the 'closest' face (i.e. with max_faces = 1)

If an application wishes to find all faces which are intersected by a ray (i.e. max_faces > 1) and requires both forward and backward facing faces, the value of 'ignore_back_faces' must be set to 'PK_LOGICAL_false'.

near and far planes
By default, the picking ray is defined as an unbounded line passing through a point in a given direction. This can be constrained using near/far planes (orthogonal to the ray) to define half-space regions.

If a near_plane is defined, only those entities lying behind the plane can be picked. If an entity intersects the near plane, it can only be picked if its 'hit point' lies behind the near plane.

If a far_plane is defined, only those entities lying in front of the plane can be picked. If an entity intersects the far plane, it can only be picked if its 'hit point' lies in front of the near plane.

Applications should avoid defining near/far planes which intersect face and edge occurrences as this may yield (apparently) inconsistent results, where some of the edges and faces which intersect the near/far planes cannot be picked.

curve tolerance and approximation
The pick tolerance values control the approximation a curve by a sequence of straight line chords. This polyline representation of a curve is used when measuring distances to edges and checking whether the ray intersects a face. When 'pick_approx' option is set to 'PK_pick_approximate_no_c' the accurate geometry instead of polyline representation is used. This method is slower. The default value of 'pick_approx' is 'PK_pick_approximate_yes_c' and it is recommended for all interactive picking operations.

The tolerance values are defined by pair of PK_LOGICAL_t and double values.
The logical denotes whether the associated double defines a tolerance value.

If picking is being performed interactively, using rendered data which has
been generated by Parasolid, the curve tolerance arguments should be set to
the same values as used when the PK rendering functions were called. This
will allow Parasolid to pick entities using the same approximate polyline
form of curves as was used to construct the drawing.

In other cases, the curve tolerance arguments can be left as default values.

method
The 'method' argument is required when picking edges and vertices.
This determines how the returned edges/vertices are ordered.

- the method argument is not used when picking faces. These will always
  be returned ordered by hit point, as if using 'PK_BODY_pick_axial_c'.

The possible settings for the 'method' argument are:

- PK_BODY_pick_radial_c
  edges/vertices are ordered by hit distance measurement

- PK_BODY_pick_axial_c
  edges/vertices are ordered by hit point (in the ray's axial direction)

- PK_BODY_pick_ratio_c'
  edges/vertices are ordered by "pick distance" (using a combination
  of the PK_BODY_pick_radial_c and PK_BODY_pick_axial_c methods)

  The value of 'ratio' is used to vary the relative weightings given to
  the radial and axial methods (its value must be positive or zero):

  if ratio > 1.0 : pick_distance = hit_distance   + rel_distance/ratio
  if ratio = 1.0 : pick_distance = hit_distance   + rel_distance
  if ratio < 1.0 : pick_distance = hit_distance*ratio + rel_distance

  The 'rel_distance' values for a set of entities E1, E2, E3 ...
  are found by ordering the associated hit points P1, P2, P3 ...
  selecting the 'front-most' point Pf
  and calculating dist(Pf, P1), dist(Pf, P2), dist(Pf, P3) ...
  where dist(Pi, Pj) is the distance from point Pi to Pj

  If ratio is large, the effect is the same as using PK_BODY_pick_radial_c
  If ratio is zero , the effect is the same as using PK_BODY_pick_axial_c

The PK_BODY_pick_topols_o_m macro initialises method to 'PK_BODY_pick_axial_c'.
This means that if edges and/or vertices are being picked at the same time as
faces, the returned entities will all be ordered by hit point. Typically this
means that entities are ordered by distance from an application's eye-point.

The 'PK_BODY_pick_radial_c' method is appropriate when firing a ray through a
'wire frame' representation of a body, where the 'hit-distance' from the ray
to candidate edges & vertices is more important than their relative distance
from the application's eye-point.

geometry classes
The option geom_classes allows the user to specify the classes of geometry
attached to topology that may be picked. The class types are passed in as an
array. At present only the surface classes for faces can be specified.

PK_BODY_pick_topols_r_t

```
struct PK_BODY_pick_topols_r_s
    {
    int                     n_faces;
    PK_BODY_pick_topol_r_t *faces;
    int                     e_faces;

    int                     n_edges;
    PK_BODY_pick_topol_r_t *edges;
    int                     e_edges;

    int                     n_vertices;
    PK_BODY_pick_topol_r_t *vertices;
    int                     e_vertices;
    };
typedef struct PK_BODY_pick_topols_r_s PK_BODY_pick_topols_r_t;
```

Structure returning picked entities

Each picked entity is recorded in a PK_BODY_pick_topol_r_t structure.

The 'entity' field refers to a picked face, an edge or a vertex.

The 'occurence' field is a number in the range [0..n_bodies-1]. It identifies an index into the owning body array and possibly an associated transformation.

If faces are being picked, the 'intersect' point is the exact intersection between the ray and face. The 'distance' values of picked faces are set to zero. If faces are picked within curve tolerance distance of their edge or silhouette boundary, the function may fail to pick the correct face.

If edges are being picked, the 'intersect' field is a 'hit point' on the ray at which the distance to the (transformed) edge occurrence was at a minimum. The 'distance' field records the hit-distance, measured from here to the edge.

If vertices are being picked, the 'intersect' field is a 'hit point' on the ray at which the distance to the (transformed) vertex occurrence was at a minimum. The 'distance' field records the distance from here to the vertex.

The arrays of picked entity arrays are returned in a PK_BODY_pick_topols_r_t structure. If no entities of a particular type were requested or if none were picked by the ray, each array is returned empty; its PK_BODY_pick_topol_r_t pointer set to NULL and its e_<entities> and n_<entities> counts set to zero.

If more than max_<entity> entities would be picked by the ray, details of the closest 'max_<entities>' are returned in the array and the number of 'excess' entities which also met the pick criteria recorded in the 'e_<entity>' count.

The excess count hints to the application that the model is topologically complex around the region of the ray or possibly that geometric selection criteria set by the application are too lax.

The returned pick data must not be used to simulate "closest approach" or "boolean" operations. This is because returned hit point and hit distance measurements may only be computed to the accuracy of the supplied (or the implied) curve tolerances, not to modelling accuracy.

The function PK_BODY_pick_topols_r_f may be used to free associated memory.

PK_BODY_problem_group_array_t

```
struct PK_BODY_problem_group_array_s
    {
    PK_BODY_problem_group_t  *array;
    int                       length;
    };

typedef struct PK_BODY_problem_group_array_s PK_BODY_problem_group_array_t;
```

A structure to embody an array of structures of type PK_BODY_problem_group_t

PK_BODY_problem_group_t

```
struct PK_BODY_problem_group_s
    {
    PK_BODY_sewing_problem_t     type;
    PK_EDGE_t                   *edges;
    int                          n_edges;
    };

typedef struct PK_BODY_problem_group_s PK_BODY_problem_group_t;
```

A structure collecting a set of edges associated with an encountered sewing problem, together with a token identifying the type of problem (if known.)

This structure embodies information on a group of edges encountered during a sewing operation that could not be successfully processed. The problem may be due to options set that prohibited the processing, or an internal limitation that prevented selection of a subset of the edges that could be glued together.

The structure has three user fields:

type: A token of type PK_BODY_sewing_problem_t that can be one of ...

PK_BODY_sewing_overlapping_c	The group of edges represent the boundaries of an area where sheets overlap to an extent too large to permit glueing them together
PK_BODY_sewing_non_manifold_c	The group of edges represent the boundaries where a collection of more than two sheets meet along a 'common' edge
PK_BODY_sewing_non_oriented_c	The group of edges are those that if sewn together would introduce a twist into the resulting sheet body.
PK_BODY_sewing_unspecified_c	The group of edges are where the algorithm couldn't decide how to sew them up and wasn't able to figure out what the problem was

n_edges: The number of edges associated with the problem
edges: An array of those edges.

PK_BODY_profile_match_t

```
struct PK_BODY_profile_match_s
    {
    PK_BODY_t                 profile;
    int                       n_matches;
    PK_BODY_loft_one_match_t *matches;
    };

typedef struct PK_BODY_profile_match_s PK_BODY_profile_match_t;
```

This structure holds an array of matches, each of which describes an association between a vertex of the given profile of the loft and one on the immediately following profile.

The matching structure has the following fields :

'profile' The profile for which this is a mapping of the vertices onto those of the next profile.

'n_matches' The number of matches held in the structure. This must not exceed the number of vertices on the profile.

'matches' An array of matches, each of which describes an association between a vertex of the given profile of the loft and one on its immediate following neighbour.

PK_BODY_section_o_t

```
struct PK_BODY_section_o_s
    {
    int                       o_t_version;     --- version number
    PK_section_fence_t        fence;           --- (PK_section_fence_both_c)
    const PK_boolean_match_o_t
                             *matched_region;  --- (NULL)
    PK_LOGICAL_t              merge_imprinted; --- (PK_LOGICAL_false)
    PK_LOGICAL_t              merge_new_faces; --- (PK_LOGICAL_false)
    PK_LOGICAL_t              selective_merge; --- (PK_LOGICAL_false)
    PK_section_check_fa_t     check_fa;        --- (PK_section_check_fa_yes_c)
    double                    default_tol;     --- (1.0E-5)
    double                    max_tol;         --- (0.0)
    PK_LOGICAL_t              tracking;        --- (PK_LOGICAL_false)
    PK_LOGICAL_t              keep_target_edges;  --- (PK_LOGICAL_false)
    };
typedef struct PK_BODY_section_o_s  PK_BODY_section_o_t;
```

This option structure is supplied to a section operation for bodies.

Description of fields:

'fence' To determine which resultant bodies or faces are returned when performing a body sectioning operation. Permitted values are :
- PK_section_fence_front_c
- PK_section_fence_back_c
- PK_section_fence_both_c

'matched_region' Describes matched regions between the target face and the tool sheet.

'merge_imprinted' Merge all mergeable imprinted edges created and/or located by the section operation.
Note the following :
- o an existing edge can be considered an imprinted edge if it forms part of the intersection between

faces from the target and tool.
o mergeable edges which are adjacent to new section faces are not merged (this behaviour is particular to sectioning).

'merge_new_faces' Merge all mergeable imprinted edges created by the section operation including edges which are adjacent to new section faces. This flag has no effect if the 'merge_imprinted' field is not set to PK_LOGICAL_true.

'selective_merge' If this is set to PK_LOGICAL_true, the merge phase will avoid mergeing imprinted edges that existed and were mergeable before the start of the section. This flag has no effect if 'merge_imprinted' is not set to PK_LOGICAL_true.

'check_fa' The value of this determines if face checking is performed during the operation, permitted values are :

PK_section_check_fa_no_c
PK_section_check_fa_yes_c

The default is set to 'PK_section_check_fa_yes_c' and it is strongly advised not to change this value. If the face checking is switched off, the section operation may produce an invalid body without raising an error.

'default_tol' The default tolerance to which the sectioning operation, if it has to, may approximate entities or assume that entities are coincident. This value should be set to a value that is related to the tolerance to which the application is creating approximations.
If this field is set to zero (0.0), it is assumed that the decision to set tolerances on entities is being left to the modeller.

'max_tol' This value specifies the maximum tolerance which may be applied to any entity involved in the sectioning operation.
If this field is set to zero (0.0), the default, then it assumed that no upper limit has been set on the tolerance of any entity.
If the field is not set to zero (0.0), then it must be set to a value that is greater than the value of the 'default_tol' field.

'tracking' Not currently in use.

'keep_target_edges' This option controls which edge survives when a target edge and a tool edge coincide in a boolean. If both faces adjacent to the tool edge survive, then the tool edge will also survive. Otherwise, the result depends on the setting of the option:

PK_LOGICAL_true:
The target edge survives, unless the target edge is a new edge imprinted during the boolean, and the tool edge is an original edge - in this case, the tool edge survives.

PK_LOGICAL_false (the default):
The older edge (i.e. the one with the smaller tag) survives.

This option only has an effect if general topology is disabled. If it is enabled, then the target edge will always survive, even if it is a newly imprinted edge.

PK_BODY_sew_bodies_o_t

```
struct PK_BODY_sew_bodies_o_s
    {
    int                      o_t_version;
    PK_LOGICAL_t             set_global_tolerance, --- (PK_LOGICAL_false)
                             allow_disjoint_result,--- (PK_LOGICAL_false)
                             treat_as_manifold;    --- (PK_LOGICAL_true)
    PK_BODY_sewing_type_t    prefered_body_type;   --- (PK_BODY_sewing_any_c)
    PK_BODY_sewing_removal_t duplicate_removal;
                                        --- (PK_BODY_sewing_remove_none_c)
    int                      number_of_iterations; --- (1)
    double                  *iteration_bounds;     --- (NULL)
    };

typedef struct PK_BODY_sew_bodies_o_s PK_BODY_sew_bodies_o_t;
```

Holds optional controls on the execution of sewing.

The option structure defines options applicable to the sewing together of sheet bodies.

The structure has six user fields:

allow_disjoint_result: If true sewing can return several disjoint sheets produced by sewing. When this is false the only sheet returned is the one produced by sewing sheets onto the first one provided in the list of received bodies.
(default is false)

treat_as_manifold: Assume the result is manifold. If have more than two edges paired, assume the closest pairing is the intended one.

prefered_body_type: An "enum" indicating the prefered type of the result.

duplicate_removal: An "enum" indicating effort to be placed in removing duplications in the input sheets.

number_of_iterations: This specifies the number of gap width bound values to iterate over while sewing. This must exceed 0.

Where the 'number_of_iterations' is 1, the gap width bound specified in the 'gap_width_bound' argument is used for a single sewing attempt.

Where the value of this option field exceeds 1, sewing will make the specified number of attempts to complete the sewing operation.

If iteration is requested and the array held in 'iteration_bounds' is not NULL, an attempt to sew the bodies will be carried out at each gap width bound specified in the array (assumed to hold 'number_of_iteration' values'). The gap width bounds will be used in the order in which they appear in the array. It is expected that the best

results will occur when the values are in ascending order.

Iteration may be requested, with the value of 'iteration_bounds' set to NULL. In this case Parasolid will generate 'number_of_iterations' ascending values terminating in 'gap_width_bound'. The range of values chosen will not exceed two orders of magnitude, other than that the chosen values are not specified.

iteration_bounds: If the number of iterations exceeds 1 this array will be used as the source of gap width bounds for each iteration unless it is set to NULL. In the latter case Parasolid will generate a range of gap width bounds based on the value of the 'gap_width_bound' passed to PK_BODY_sew_bodies.

Where the number of iterations is one, this option field is ignored.

PK_BODY_sewing_problem_t

typedef int PK_BODY_sewing_problem_t;

Enum to indicate type of problem encountered in sewing PK_BODY_sew_bodies

PK_BODY_sewing_overlapping_c - found sheets overlapping along given edges
PK_BODY_sewing_non_manifold_c - more than two sheets meet at given edges
PK_BODY_sewing_non_oriented_c - sewing given edges would twist sheet once
PK_BODY_sewing_unspecified_c - other internal failure encountered

PK_BODY_sewing_removal_t

typedef int PK_BODY_sewing_removal_t;

Enum to indicate level of effort to remove duplications from input sheets to PK_BODY_sew_bodies

PK_BODY_sewing_remove_none_c - Don't bother at all

PK_BODY_sewing_remove_poss_c - Identify possible duplicates to within modeller precision.

PK_BODY_sewing_remove_cert_c - Identify, with reasonable certainty, sheets duplicated to within the tolerance specified by the gap width bound passed to PK_BODY_sew_bodies.

PK_BODY_sewing_remove_cert_c will be slower and more successful in identifying duplicates than PK_BODY_sewing_remove_poss_c due to a greater effort made in comparing sheets.

PK_BODY_sewing_type_t

typedef int PK_BODY_sewing_type_t;

Enum to indicate desired body type of the results from applying PK_BODY_sew_bodies

```
PK_BODY_sewing_solid_c   - make closed sheets into solids
PK_BODY_sewing_sheets_c  - leave results as sheets unless non-manifold
PK_BODY_sewing_general_c - leave sheets as general bodies ( always )
PK_BODY_sewing_any_c     - make sheet least general possible ( i.e.
                           prefer in order: solid, sheet, general )
```

PK_BODY_simplify_t

typedef int PK_BODY_simplify_t;

This data type indicates the level of simplification to be undertaken when producing swept or lofted bodies. It applies to the local simplification of the surfaces generated by the sweep or loft, but will not affect the topology of the resulting body.

It has two values

PK_BODY_simplify_yes_c	Try to make local analytic simplifications
PK_BODY_simplify_no_c	Attempt no simplications

PK_BODY_sweep_alignment_t

typedef int PK_BODY_sweep_alignment_t;

Specifies whether a profile should be swept in such a way that its normal always points in the same direction, or should remain at a fixed angle to the tangent of the path.

PK_BODY_sweep_align_normal_c	Keep normal at fixed angle to path tangent
PK_BODY_sweep_align_parallel_c	Keep normal at fixed angle to global axes

PK_BODY_sweep_fair_t

typedef int PK_BODY_sweep_fair_t;

Specifies what method (if any) should be used to attempt to improve the method of sweeping along the supplied path.

PK_BODY_sweep_fair_no_c	The method of sweeping is not improved.
PK_BODY_sweep_fair_twist_c	The orientation of the profile will not twist along regions of the path which are nearly planar, or regions where the path twists rapidly.

PK_BODY_sweep_fault_t

typedef int PK_BODY_sweep_fault_t;

A status value to be associated with swept bodies. It indicates the type of problem encountered during the construction of the body (may be none at all).

PK_BODY_sweep_ok_c	Swept body created without problems
PK_BODY_sweep_failure_c	Internal algorithm failure
PK_BODY_sweep_bad_path_c	Invalid sweep path
PK_BODY_sweep_bad_profile_c	Invalid sweep profile
PK_BODY_sweep_small_profile_c	Profile is too small to sweep
PK_BODY_sweep_surf_failure_c	Failed to create swept surface
PK_BODY_sweep_curve_failure_c	Failed to create profile curve

```
    PK_BODY_sweep_geom_self_int_c  Swept body has self intersection in at least
                                   one of its geometries.
    PK_BODY_sweep_geom_degen_c     Swept body has degeneracy in at least
                                   one of its geometries.
    PK_BODY_sweep_topol_self_int_c Swept body has an invalid contact between
                                   two of its faces, edges, or vertices.
    PK_BODY_sweep_check_failure_c  Fault detected in generated body
```

PK_BODY_sweep_law_discrete_t

```
struct PK_BODY_sweep_law_discrete_s
    {
    int              n_vertices;
    PK_VERTEX_t     *vertices;
    double          *values;
    };

typedef struct PK_BODY_sweep_law_discrete_s PK_BODY_sweep_law_discrete_t;
```

This structure holds a discrete description of a scalar law curve for twist or scale in sweeping as a set or vertex/value pairs.

The law description structure has the following fields :

'n_vertices' The number of vertices with specified law values

'vertices' An array of the vertices that have specified law values.

'values' An array of values. Each entry is the law value for the vertex in the corresponding entry of 'vertices'

The following rules apply to the vertex/value pairs that may be specified via this structure :

- Any number of vertex/value pairs may be specified.
- The vertex/value pairs are not presumed to be in any particular order except in the case that the start and end vertices of the path are the same. Where this is true, the first occurrence of the start/end vertex is taken to be associated with the start value of the function, and the second occurrence is associated with the end value.

The following points just describe two ways in which the swept body's shape can be controlled via these twist or scale controls :

- Constant Twist or Scaling
  The user can affect a constant twist or scaling of the sweep by specifying a single vertex/value pair. The specified vertex maybe any vertex belonging to the sweep path profile. The twist or scaling transformation is applied to the whole sweep.
- Linear Interpolation Between Start and End
  The user can affect a linear interpolated twist or scaling over the length of the sweep path by just specifying the start and end vertices within two vertex/value pairs. The start and end vertex/value pairs can be specified in any order.

PK_BODY_sweep_law_t

```
struct PK_BODY_sweep_law_s
    {
    PK_BODY_sweep_law_type_t        law_type;
    PK_BODY_sweep_law_discrete_t    law_set;
    PK_LAW_sf_t                     law_curve;
    };

typedef struct PK_BODY_sweep_law_s PK_BODY_sweep_law_t;
```

This structure holds the description of a scalar law curve for twist or scale in sweeping.

The law description structure has the following fields :

'law_type'	Specifies whether the description is empty (PK_BODY_sweep_law_none_c) or has a discrete vertex/value description (PK_BODY_sweep_law_discrete_c) or a functional description (PK_BODY_sweep_law_curve_c).
'law_set'	Filled in only if the 'law_type' indicates that the description is discrete. Holds a set of vertex/value associations
'law_curve'	Filled in only if the 'law_type' indicates that the description is functional. Holds a one-dimensional law described by a PK_LAW_sf_t standard form.

PK_BODY_sweep_law_type_t

```
typedef int PK_BODY_sweep_law_type_t;
```

The value of this token indicates the method by which a twist or scale law is being specified through the PK

PK_BODY_sweep_law_none_c	The law structure is empty/unused.
PK_BODY_sweep_law_discrete_c	The law is specified by giving a discrete set of values to be interpolated at the vertices of the path
PK_BODY_sweep_law_curve_c	The law is specified by a 1-dimensional B-spline curve parameterised [0 1]. The values will be applied in arc-length proportional method along the length of the path.

PK_BODY_sweep_status_r_t

```
struct PK_BODY_sweep_status_s
    {
    PK_BODY_sweep_fault_t       fault;
    PK_VECTOR_t                 fault_location;
    PK_TOPOL_t                  fault_topol;
    };

typedef struct PK_BODY_sweep_status_s PK_BODY_sweep_status_r_t;
```

A structure packaging status and error location information

The structure has the following fields :

'fault' A token indicating the degree of success in completing the swept body. Note that this may indicate failure even where PK_BODY_make_swept_body/profiles returns PK_ERROR_no_errors.

'fault_location' The three space point where the error is located, or from where it originated.

'fault_topol' The topology on which the error is located, or from which it originated.

PK_BODY_t

This type represents a BODY. A body is the basic modelling unit and typically corresponds to a component.

An body is a type of part (PK_CLASS_body is a subclass of PK_CLASS_part).

PK_BODY_taper_o_t

```
struct PK_BODY_taper_o_s
    {
    int                       o_t_version;      --- version number of option
                                                --- structure
    double                    tolerance;        --- max. tol. for applying
                                                --- surfaces (0.00001)
    PK_LOGICAL_t              miter_at_parting; --- whether to miter halves
                                                --- at the parting sheet
                                                --- (PK_LOGICAL_true)
    PK_LOGICAL_t              merge_face;       --- whether tapered faces
                                                --- are to be merged with
                                                --- their neighbours
                                                --- (PK_LOGICAL_true)
    PK_check_fa_fa_t          check_fa_fa;      --- check for face-face
                                                --- inconsistencies
                                                --- (PK_check_fa_fa_yes_c)
    PK_taper_method_t         default_method;   --- default tapering method
                                                --- (PK_taper_method_isocline_c)
    int                       n_methods;        --- number of non-default
                                                --- tapering methods (0)
    const PK_taper_method_t *methods;           --- non-default tapering
                                                --- methods (NULL)
    const PK_ENTITY_t       *method_refs;       --- reference entities to use
                                                --- non-default tapering
                                                --- method (NULL)
    };
typedef struct PK_BODY_taper_o_s PK_BODY_taper_o_t;
```

This structure contains optional controls for PK_BODY_taper

Description of fields:

'tolerance' Maximum tolerance for the operation. The default value is 0.00001

'miter_at_parting' If this flag is set to PK_LOGICAL_true, the resulting body will be mitered. The default value is PK_LOGICAL_true.

'merge_face' If this flag is set to PK_LOGICAL_true, the new faces produced by the operation will be

```
                            merged with their neighbouring faces, if it is
                            feasible. The default value is PK_LOGICAL_true.

'check_fa_fa'               Check for face-face inconsistencies.
                            Values can be :
                                PK_check_fa_fa_no_c
                                PK_check_fa_fa_yes_c
                            The default value is PK_check_fa_fa_yes_c.

'default_method'            Tapering method to construct new surfaces.
                            Allowed values are:
                                PK_taper_method_isocline_c : isocline method
                                PK_taper_method_curve_c    : curve method
                                PK_taper_method_surface_c  : surface method

                            The default is PK_taper_method_isocline_c.

'n_methods'                 Number of elements in 'methods' and 'method_refs'
                            arrays. Default is 0.

'methods'                   Array of 'n_methods' tapering methods to be
                            used instead of 'default_method' corresponding
                            to reference entities in 'method_refs'. Default is
                            NULL.

'method_refs'               Array of reference entities to use non-default
                            tapering method specified in corresponding
                            entries of 'methods'. Only entities given as
                            references to PK_BODY_taper are allowed in this
                            array.
```

PK_BODY_thicken_o_t

```
struct PK_BODY_thicken_o_s
    {
    int                      o_t_version;   --- version number
    int                      n_faces;       --- number of non-default faces
                                            --- (0)
    const int               *faces;         --- faces without default offsets
                                            --- (NULL)
    const double            *front_offsets; --- front offsets
                                            --- (NULL)
    const double            *back_offsets;  --- back offsets
                                            --- (NULL)
    PK_check_fa_fa_t         check_fa_fa;   --- check for face-face
                                            --- (PK_check_fa_fa_yes_c)
    PK_thicken_method_t      method;        --- thickening method
                                            --- (PK_thicken_method_offset_c )
    PK_VECTOR1_t             punch_dir;     --- punch direction
                                            --- (zero vector)
    PK_offset_method_t       offset_method; --- offset method
                                            --- (PK_offset_method_accurate_c )
    };

typedef struct PK_BODY_thicken_o_s PK_BODY_thicken_o_t;
```

This option structure allows individual faces to be given offsets different from the defaults, and controls what checking is performed.

```
Description of fields:

'n_faces'           Length of 'faces', 'front_offsets' and 'back_offsets
                    arrays.
```

'faces'	Array of length 'n_faces', containing faces with non-default offsets.
'front_offsets'	Array of length 'n_faces', of front offsets in correspondence to the face array.
'back_offsets'	Array of length 'n_faces', of back offsets in correspondence to the faces array.
'check_fa_fa'	Check for face-face inconsistencies. Permitted values are: PK_check_fa_fa_yes_c PK_check_fa_fa_no_c
'method'	Thickening method. Permitted values are: PK_thicken_method_offset_c: The side surfaces are created by ruling the side edges along the direction of the surfaces normals. PK_thicken_method_punch_c: The side surfaces are created by sweeping the side edges along an user supplied punch direction. The default value is: PK_thicken_method_offset_c.
'punch_dir'	Vector of the punch direction. This must be set if 'method' is PK_thicken_method_punch_c, and otherwise is ignored. The default value is a zero vector.
'offset_method'	The method used for offsetting surfaces in the body. The default value is PK_offset_method_accurate_c.

PK_BODY_thicken_r_t

typedef PK_TOPOL_local_r_t PK_BODY_thicken_r_t;

PK_BODY_topology_t

typedef int PK_BODY_topology_t;

This data type specifies how bodies created by lofting and sweeping will be divided into faces.

It has three values

PK_BODY_topology_minimal_c	Produce the minimum number of faces (default).
PK_BODY_topology_columns_c	Produces columns of faces corresponding to the profile edges. For sweeping, the result is a column of faces for each edge in the profile. For lofting, the result is a face for each set of corresponding edges in the profiles.
PK_BODY_topology_grid_c	Produces a grid of faces. For sweeping the result is a set of faces for each edge in the profile; the number of faces in the sweep direction is the number of edges in the path. For lofting the result is a set of faces for each set of corresponding edges in the profiles; the number of faces in the loft direction is one less than the number of profiles.

PK_BODY_tracked_loft_r_t

```
struct PK_BODY_tracked_loft_s
    {
    PK_BODY_loft_status_r_t      status;
    PK_BODY_t                    body;
    PK_TOPOL_track_r_t           tracking_info;
    };

typedef struct PK_BODY_tracked_loft_s PK_BODY_tracked_loft_r_t;
```

The structure packages a single lofted result body along with its associated status, and tracking information.

The structure has the following fields :

'status' A structure holding a status token and location information on some problem arising during the attempt to loft the profiles.

'body' The lofted body produced by PK_BODY_make_lofted_body. It may not check or be valid in cases where the status indicates anything other than complete success.

'tracking_info' A collection of structures that hold associations between the topology presented to PK_BODY_make_lofted_body and those present in the body returned in the 'body' field.

The function PK_BODY_tracked_loft_r_f may be used to free associated memory.

PK_BODY_tracked_sweep_r_t

```
struct PK_BODY_tracked_sweep_r_s
    {
    PK_BODY_sweep_status_r_t     status;
    PK_BODY_t                    body;
    PK_TOPOL_track_r_t           tracking_info;
    };

typedef struct PK_BODY_tracked_sweep_r_s PK_BODY_tracked_sweep_r_t;
```

A structure packaging a single swept result body along with its associated status, and tracking information.

The structure has the following fields :

'status' A token indicating the degree of success in completing the swept body. Note that this may indicate failure even where PK_BODY_make_swept_body returns PK_ERROR_no_errors.

'body' The swept body produced by PK_BODY_make_swept_body. It may not check or be valid in cases where the status indicates anything other than complete success.

'tracking_info' A collection of structures that hold associations between the topology presented to PK_BODY_make_swept_body and those present in the body returned in the 'body' field.

The function PK_BODY_tracked_sweep_r_f may be used to free associated memory.

PK_BODY_transform_o_t

```
struct PK_BODY_transform_o_s
    {
    int              o_t_version;  --- version number of option structure
    PK_LOGICAL_t     merge_face;   --- whether transformed faces are to be
                                   --- merged with their neighbours,
                                   --- if feasible (PK_LOGICAL_true)
    PK_check_fa_fa_t check_fa_fa;  --- check for face-face inconsistencies
                                   --- (PK_check_fa_fa_yes_c)
    };
typedef struct PK_BODY_transform_o_s PK_BODY_transform_o_t;
```

This structure contains optional controls for PK_BODY_transform_2.

Description of fields:

'merge_face' — If this flag is set to PK_LOGICAL_true, the new faces produced by the operation will be merged with their neighbouring faces, if it is feasible. The default value is PK_LOGICAL_true.

'check_fa_fa' — Check for face-face inconsistencies.
Values can be :
    PK_check_fa_fa_no_c
    PK_check_fa_fa_yes_c
The default value is PK_check_fa_fa_yes_c.

PK_BODY_type_t

```
typedef int PK_BODY_type_t;
```

PK_BODY_type_minimum_c	a topologically zero dimensional manifold body containing a single isolated vertex.
PK_BODY_type_acorn_c	a topologically zero dimensional manifold body containing two or more isolated vertices.
PK_BODY_type_wire_c	a topologically one dimensional manifold body containing one or more connected sets of edges, where any vertex is at the junction of no more than two edges.
PK_BODY_type_sheet_c	a topologically two dimensional manifold body containing one or more connected sets of faces, where any edge is at the junction of no more than two faces.
PK_BODY_type_solid_c	a topologically three dimensional manifold body containing one or more disjoint and separate solid regions. All faces form a boundary between a solid and a void region.
PK_BODY_type_general_c	a body which is non-manifold and/or of mixed topological dimensionality

PK_boolean_check_fa_t

```
typedef int PK_boolean_check_fa_t;
```

PK_boolean_check_fa_t has the following values:

```
    PK_boolean_check_fa_no_c          do not perform any face checks
    PK_boolean_check_fa_yes_c         perform individual face checks
```

PK_boolean_config_o_t

```
struct PK_boolean_config_o_s
    {
    int             o_t_version;        --- version number
    PK_LOGICAL_t    no_tool_intersect;  --- (PK_LOGICAL_false)
    PK_LOGICAL_t    no_loop_intersect;  --- (PK_LOGICAL_false)
    PK_LOGICAL_t    identical_intersect;--- (PK_LOGICAL_true)
    PK_LOGICAL_t    one_in_all_in;      --- (PK_LOGICAL_true)
    };
typedef struct PK_boolean_config_o_s PK_boolean_config_o_t;
```

This option structure describes the configuration of the tool and target bodies.

Description of fields:

'no_tool_intersect' Surviving regions of one tool body do not intersect with those of another tool body.

'no_loop_intersect' Tool bodies do not intersect target body edges.

'identical_intersect' The intersection loops of each tool body with the target body are identical with those of all other tool bodies.

'one_in_all_in' If one tool intersects a target face then all tools intersect that target face.

PK_boolean_fence_t

```
typedef int PK_boolean_fence_t;
```

PK_boolean_fence_t has three values:

```
    PK_boolean_fence_front_c          keep bodies in front
    PK_boolean_fence_back_c           keep bodies behind
    PK_boolean_fence_none_c           keep all bodies
```

PK_boolean_function_t

```
typedef  int PK_boolean_function_t;
```

PK_boolean_function_t has three values:

```
    PK_boolean_intersect_c        intersect target and tool entities
    PK_boolean_subtract_c         subtract tool entity from target entity
    PK_boolean_unite_c            unite target and tool entities
```

PK_boolean_match_o_t

```
struct PK_boolean_match_o_s
    {
    int                       o_t_version;     --- version number
    int                       n_match_regions; --- (0)
    const PK_boolean_match_region_o_t *match_regions;   --- (NULL)
    PK_LOGICAL_t              auto_match;      --- (PK_LOGICAL_false)
```

```
    double                    auto_match_tol;  --- (0.0)
    PK_boolean_match_style_t  match_style; --- (PK_boolean_match_style_basic_c)
    };
typedef struct PK_boolean_match_o_s PK_boolean_match_o_t;
```

This option structure is used to specify matched regions of the target and tool body.

Matched topological regions may be passed to the boolean functions as a means of aiding and/or enhancing the performance of the required boolean operation.

Matched topology can only be specified for boolean operations between manifold bodies.

Refer to 'PK_boolean_match_region_o_t' documentation for a full description of matched regions.

Description of fields:

'n_match_regions'	The number of matched region structures supplied.
'match_regions'	Array of structures describing the matched topological regions.
'auto_match'	Attempt to automatically locate matched faces in the target and tool bodies. Automatic matching is performed once the matched regions specified within 'match_regions' have been processed. If 'match_style' is PK_boolean_match_style_basic_c then only faces with planar geometry are considered for automatic matching. All faces are considered if 'match_style' is PK_boolean_match_style_relax_c.
'auto_match'	This option is not currently in use.
'auto_match_tol'	The tolerance used to automatically detect matched topology.
'match_style'	If 'match_style' is PK_boolean_match_style_basic_c then the application must ensure that matched items do not interact other than in the supplied topology. Parasolid will look for other interactions if 'match_style' is set to PK_boolean_match_style_relax_c.

PK_boolean_match_region_o_t

```
struct PK_boolean_match_region_o_s
    {
    int                   o_t_version; --- version number
    PK_TOPOL_t            regions[2];  --- (PK_ENTITY_null, PK_ENTITY_null)
    PK_boolean_match_t    match_type;  --- (PK_boolean_match_exact_c)
    double                tolerance;   --- (0.0)
    };
typedef struct PK_boolean_match_region_o_s PK_boolean_match_region_o_t;
```

This option structure is used to specify and describe matched regions of the target and tool body.

This options structure provides a mechanism to inform the boolean functions of known matched topological regions. A matched region is a region of overlap between two topological entities which are geometrically coincident to a specified tolerance. The entities in the matched region must be in the target or tools of the boolean operation.

The following combinations of topological entities may be provided as matched regions :

- o Vertex - Vertex
- o Edge - Edge
- o Face - Edge
- o Face - Face

The matched entities or regions are associated with a 'match_type' which classifies the relationship between the entities :

Vertex-Vertex Regions

o Exact : Invalid classification.

o Contains : Invalid classification.

o Overlap : Invalid classification.

o Imprinted : The vertices are matched due to being paired via an imprinting operation.

Edge-Edge Regions

o Exact : The edge curves are coincident to within the specified tolerance. Equivalent edge vertices are coincident to within the specified tolerance.

o Contains : The edge curves are coincident to within the specified tolerance. The bounds of the second edge lie completely within the first edge.

o Overlap : The edge curves are coincident to within the specified tolerance. There is at least some form of overlap between the two edges.

o Imprinted : The edges are matched due to being paired via an imprinting operation.

Face-Edge Regions

o Exact : Invalid classification.

o Contains : The edge curve is coincident to within the specified tolerance of the face surface. The edge lies completely within the face. The edge may be coincident with an edge of the face.

o Overlap : The edge curve is coincident to within the specified tolerance of the face surface. The edge may intersect the boundary (edges) of the face. The edge may be coincident with an edge of the face.

o Imprinted : Invalid classification.

Face-Face Regions

o Exact : The face surfaces are coincident to within the specified tolerance. For every edge on the first face there is a matching edge on the second face. The edge matches are not required to be exact.

o Contains : The face surfaces are coincident to within the specified tolerance. The second face lies completely within the first face. None of the edges of the second face may intersect the boundary of the first face. Although edges of the second face are allowed to be coincident with those of the first face.

o Overlap : The face surfaces are coincident to within the specified tolerance. The boundary of one face may intersect the

boundary of the other face. Edges of one face may be coincident with edges of the other face.

o Imprinted : Invalid classification.

Matched Region Operation

o The topological entities supplied with a matched region may only consist of vertices, edges or faces.

o The topological entities supplied with a matched region must belong to different bodies. Matched regions between two tool bodies is permitted.

o The matched regions may be specified in any order. Internally, the matched regions are sorted into groups of vertex-vertex, edge-edge, face-edge and face-face regions. The matched regions are then processed in this order. The order of processing within a group depends on the input order.

o A matched region may be specified more than once :
  o Directly : For example, two edge-edge matches may be input, where the edges specified are the same. The first occurence of the matched region will be processed and the second will be ignored.

  o Indirectly : For example, if two matched regions are specified, an edge-edge and a face-face. The edges may form part of the boundary of the faces. The order of processing is dictated by the internal ordering of the matched regions, described above. In this case, the edge-edge match will be processed first and then the face-face match; the face-face match will ignore the specified edges when they are encountered.

o Edges may be 'tolerantised' as a result of the matching operation.

Matched Region Assumption

The specification of matched regions ultimately specifies an edge (or two edges) which forms the intersection between a number of faces. If the 'match_style' field in the PK_boolean_match_o_s structure is PK_boolean_match_style_basic_c then the implementation of matched regions assumes that the edge (or edges) specifies a unique intersection between all the faces that meet at that edge (or edges). The faces that are adjacent to the edges involved in the matched region are not tested for intersection. This means, if the faces intersect elsewhere, those intersections will not be detected (unless those intersections are specified by additional matched regions).

This assumption is not made if 'match_style' is set to PK_boolean_match_style_relax_c and Parasolid will find any further interactions.

Description of fields:

'regions' The matched topological entities. Only vertices, edges and faces are considered valid topological entities for matching.

'match_type' Classifies the topological match, permitted values are,

PK_boolean_match_exact_c
PK_boolean_match_contains_c
PK_boolean_match_overlap_c
PK_boolean_match_imprinted_c

'tolerance' The value of the tolerance specifies the maximum deviation between the matched topological entities in the region of overlap.

PK_boolean_match_style_t

```
typedef int PK_boolean_match_style_t;
```

PK_boolean_match_style_t currently has two values:

PK_boolean_match_style_basic_c	original match specification
PK_boolean_match_style_relax_c	relax the match conditions

PK_boolean_match_t

```
typedef int PK_boolean_match_t;
```

PK_boolean_match_t has four values:

PK_boolean_match_exact_c	regions match exactly
PK_boolean_match_contains_c	first region contains second region
PK_boolean_match_overlap_c	regions overlap
PK_boolean_match_imprinted_c	regions have been imprinted

PK_boolean_r_t

```
struct PK_boolean_r_s
    {
    PK_boolean_result_t        result;
    int                        n_bodies;
    PK_BODY_t                 *bodies;
    int                        n_reports;
    PK_boolean_report_r_t     *reports;
    };
typedef struct PK_boolean_r_s PK_boolean_r_t;
```

This data structure contains operation status and the boolean results.

Description of fields:

'result' This status field indicates the overall state of the boolean operation.

'n_bodies' The length of the 'bodies' array.

'bodies' The array of resultant bodies from the boolean operation. If the 'result' field is set to 'PK_boolean_result_fail_c', the array will be set to null.

'n_reports' The number of boolean reports in the 'reports' array.

'reports' This array may contain supplementary information regarding the outcome of the boolean operation. For example, if the operation failed, the reports may contain topology related to the failure.

Destroying the returned data structure

When the application wishes to free the memory associated with the results structure, the function 'PK_boolean_r_f' must be used.

PK_boolean_region_select_o_t

```
struct PK_boolean_region_select_o_s
    {
    int                          o_t_version;       --- version number
    PK_boolean_region_t          select_type;       --- (PK_boolean_include_c)
    int                          n_region_selectors;--- (0)
    const PK_TOPOL_t            *selectors;         --- (NULL)
    const PK_boolean_select_t   *selector_types;    --- (NULL)
    int                          n_help_points;     --- (0)
    const PK_VECTOR_t           *help_points;       --- (NULL)
    };
typedef struct PK_boolean_region_select_o_s PK_boolean_region_select_o_t;
```

This option structure is used to identify regions of the tool body to be included or excluded from a boolean.

A region is this context is a boundary faceset which lies wholly inside or outside the target body. (Tool faces which intersect the target body have been split by the boolean: the resultant faces lie wholly inside or outside the target body)

Each region is identified by a single face, edge or vertex which is interior to it; it is not possible to specify some regions for inclusion and some for exclusion, they will all be either included or excluded.

Description of fields:

'select_type' Indicates whether to include or exclude specified regions, permitted values are

PK_boolean_include_c
PK_boolean_exclude_c

'n_region_selectors' The number of topological entities supplied.

'selectors' Topological entities which each uniquely identify a region. The array may contain a mixture of faces, edges or vertices, but no other types.
The first 'n_help_points' elements of this array are associated with the elements of the 'help_points' array.
If a help point is not to be associated with any tool face, then the appropriate element of this array must be set to 'PK_ENTITY_null'.

'selector_types' Indicates the type of region selection. This array may be null or contain 'n_region_selectors' elements.
If a null array is specified, then a default type of 'PK_boolean_select_specific_c' is assumed.
Permitted values are,

PK_boolean_select_specific_c
PK_boolean_select_adjacent_c
PK_boolean_select_propagate_c

The 'PK_boolean_select_propagate_c' type is only valid for faces and edges specified as regions.

'n_help_points' The number of help points (vectors) supplied.

'help_points' Help points (vectors) which, when combined with the appropriate topological entity from the 'selectors' array, unambiguously identifies a region.
The elements of this array are associated with the first 'n_help_points' of the 'selectors' array.

Help points may only be associated with a face. If a help point is associated with 'PK_ENTITY_null', then the tool face closest to the help point is located and used as the face.

PK_boolean_region_t

```
typedef  int PK_boolean_region_t;
```

PK_boolean_region_t has the values:

PK_boolean_include_c	include regions in the result
PK_boolean_exclude_c	exclude regions from the result
PK_boolean_off_c	turns off the selection facility

PK_boolean_report_r_t

```
struct PK_boolean_report_r_s
    {
    PK_boolean_report_t        report;
    int                        n_entities;
    PK_ENTITY_t               *entities;
    };
typedef struct PK_boolean_report_r_s PK_boolean_report_r_t;
```

This data structure contains information that pertains to the result of the boolean operation.

Description of fields:

'report' This report field indicates the nature of this boolean report, permitted values are :

PK_boolean_report_failure_c
PK_boolean_report_opposed_sheets_c
PK_boolean_report_solid_has_void_c
PK_boolean_report_partial_coi_c
PK_boolean_report_t_sheet_c
PK_boolean_report_non_manifold_c
PK_boolean_report_invalid_match_c
PK_boolean_report_imprint_fail_c
PK_boolean_report_intersect_fail_c
PK_boolean_report_invalid_face_c

'n_entities' The length of the 'entities' array.

'entities' This array may contain topological or geometric entities related to the boolean report. For example, if the 'report' field is returned with 'PK_boolean_report_invalid_match_c', the 'entities' array will contain the invalid matched region.

PK_boolean_report_t

```
typedef int PK_boolean_report_t;
```

PK_boolean_report_t has the following values:

PK_boolean_report_failure_c	general failure
PK_boolean_report_opposed_sheets_c	attempt to unite opposed sheets
PK_boolean_report_solid_has_void_c	solid has illegal void
PK_boolean_report_partial_coi_c	failure due to partial coincidence
PK_boolean_report_t_sheet_c	operation would result in T-sheet

```
    PK_boolean_report_non_manifold_c     operation would result in non-manifold
                                         body
    PK_boolean_report_invalid_match_c    invalid matched region specified
    PK_boolean_report_imprint_fail_c     face imprinting operation failed
    PK_boolean_report_intersect_fail_c   face clashing operation failed
    PK_boolean_report_invalid_face_c     invalid face detected
```

PK_boolean_result_t

```
typedef int PK_boolean_result_t;

PK_boolean_result_t has the following values:
    PK_boolean_result_success_c         boolean succeeded
    PK_boolean_result_no_clash_c        target does clash with tools
    PK_boolean_result_no_effect_c       target not affected by tools
    PK_boolean_result_failed_c          boolean failed
```

PK_boolean_select_t

```
typedef int PK_boolean_select_t;

PK_boolean_select_t has three values:
    PK_boolean_select_specific_c        specified region must not be on or
                                        adjacent to imprinted topology
    PK_boolean_select_adjacent_c        specified region may be on or
                                        adjacent to imprinted topology
    PK_boolean_select_propagate_c       region selection will be propagated
                                        to new regions if the specified
                                        region is split
```

PK_bound_def_t

```
struct PK_bound_def_s
    {
    PK_bound_t        bound;        --- method of defining bound
    PK_LOGICAL_t      forward;      --- whether bound in positive sense of path
    double            distance;     --- extrusion distance
    PK_ENTITY_t       entity;       --- bounding entity
    PK_LOGICAL_t      nearest;      --- whether divisions are numbered from
                                    --- nearest to or furthest from the
                                    --- profile
    int               nth_division; --- select nth division
    };
typedef struct PK_bound_def_s PK_bound_def_t;
```

This structure describes how an extruded body or feature is bound in one direction.

It has the following fields:

'bound' — method used for defining this bound. See the documentation for PK_bound_t

'forward' — whether the bound is on the side of the given 'profile' in the direction of the given 'path' vector, or on the other side.

'distance' — distance to bound when 'bound' is PK_bound_distance_c. This must be greater than or equal to zero. To extrude in the opposite direction to the 'path' vector, 'forward' is set to PK_LOGICAL_false.

'entity' a bounding entity. This may be a body, face or surface depending upon 'bound'.

'nearest' if PK_LOGICAL_true then divisions are numbered starting from one and increasing in the direction moving away from the profile. If 'nearest' is set to PK_LOGICAL_false then the first division is that furthest from the 'profile' and division numbers increase towards the 'profile'.

'nth_division' when 'bound' is one of: PK_bound_surf_c, PK_bound_face_c, PK_bound_body_c or PK_bound_sheet_t then 'nth_division' may be set to select a particular division.

PK_bound_t

typedef int PK_bound_t;

This data type specifies how an extruded body or feature is to be bounded.

PK_bound_distance_c	the extrusion is bounded by a copy of the 'profile' at a given distance from the given 'profile'. The distance must be non-negative. Putting the bound in the opposite direction to the 'path' vector is achieved by setting 'forward' in the PK_bound_def_t structure to PK_LOGICAL_false.
PK_bound_surf_c	the extrusion is bounded by a given surface. The surface may be given explicitly, or a face may be given, in which case the face's surface will be used. The division may extend beyond the face on the face's surface. The surface is not affected by the operation.
PK_bound_face_c	the extrusion is bounded by the given face, and, if necessary, whatever adjacent faces are required to complete a division. The face and any necessary adjacent faces are not affected by the operation.
PK_bound_body_c	the extrusion is bounded by its intersection with the given body. The body is not affected by the operation. The body must be a sheet body or solid body.
PK_bound_sheet_c	the extrusion is bounded by its intersection with the given sheet body. When the profile is a wire or sheet body, the bounding sheet body is destroyed by the operation.

Note on differences between PK_bound_body_c and PK_bound_sheet_c: When the PK_bound_body_c option is given then Parasolid will find candidate faces on the sheet or solid body which may form a division; ie only some faces of the given body may be involved in the boolean operation to find the division. With the PK_bound_sheet_c option, all faces of the accompanying sheet body will be used to find divisions. Hence the difference is largely one of performance: if the application knows the relevant set of faces then it is best to use these to create a sheet body (PK_FACE_make_sheet_body) and to use the PK_bound_sheet_c option. If the application does not know which faces are likely to form the division then the PK_bound_body_c option should be used.

PK_BOX_t

```
struct PK_BOX_s
    {
    double coord[6];
    };
typedef struct PK_BOX_s PK_BOX_t;
```

This structure represents a rectangular box aligned with the world coordinate axes.

The components are as follows:

```
    'coord[0]'   minimum X
    'coord[1]'   minimum Y
    'coord[2]'   minimum Z
    'coord[3]'   maximum X
    'coord[4]'   maximum Y
    'coord[5]'   maximum Z
```

PK_BSURF_array_t

This structure contains an array of entities of type PK_BSURF_t

```
typedef struct PK_BSURF_array_s
    {
    PK_BSURF_t     *array;   --- array of B geometry SURFaces
    int             length;  --- length of array
    }
    PK_BSURF_array_t;
```

It has the fields:

'array': array of entities of type PK_BSURF_t

'length': the number of entities in 'array'

PK_BSURF_form_t

```
typedef int PK_BSURF_form_t;
```

This field can take any one of the following values.

```
    'PK_BSURF_form_unset_c'       : The surface shape has not been calculated.
    'PK_BSURF_form_arbitrary_c'   : The surface shape is not special.
    'PK_BSURF_form_planar_c'      : The surface is a piece of plane.
    'PK_BSURF_form_cylindrical_c' : The surface is a piece of cylinder.
    'PK_BSURF_form_conical_c'     : The surface is a piece of cone.
    'PK_BSURF_form_spherical_c'   : The surface is a piece of sphere.
    'PK_BSURF_form_toroidal_c'    : The surface is a piece of torus.
    'PK_BSURF_form_revolved_c'    : The surface is a b-spline curve revolved.
    'PK_BSURF_form_ruled_c'       : The surface is ruled.
    'PK_BSURF_form_gen_cone_c'    : The surface is a generalised cone.
        The above is not supported at v7.0 and will be translated as 'unset'.
    'PK_BSURF_form_quadric_c'     : The surface is quadric.
    'PK_BSURF_form_swept_c'       : The surface is a b-spline curve swept.
```

PK_BSURF_piecewise_sf_t

```
struct PK_BSURF_piecewise_sf_s
    {
    int                 u_degree;           --- The degree in u = order-1.
    int                 v_degree;           --- The degree in v = order-1.
    int                 n_u_segments;       --- The number of segments in u.
    int                 n_v_segments;       --- The number of segments in v.
    int                 dim;                --- Dimension of coefficient vecs
    PK_LOGICAL_t        is_rational;        --- True if the surface is rational
    PK_piecewise_rep_t  rep;                --- Representation method
    double             *coeffs;             --- Coefficient vectors
    };
typedef struct PK_BSURF_piecewise_sf_s PK_BSURF_piecewise_sf_t;
```

This data structure is the piecewise standard form of a b-surface.

Specific Errors

```
    PK_ERROR_bad_value             unreadable value
    PK_ERROR_bad_dimension         illegal 'dim'
    PK_ERROR_bad_order             hermite basis and degree not 3
    PK_ERROR_discontinuous_surface adjacent segments do not meet.
```

'u_degree' This field specifies the degree of the surface in the u direction. The minimum degree is 1. If the Hermite basis is used ('rep' = PK_piecewise_rep_hermite_c) then the surface must be bicubic ('degree' = 3).

'v_degree' This field specifies the degree of the surface in the v direction. The minimum degree is 1. If the Hermite basis is used ('rep' = PK_piecewise_rep_hermite_c) then the surface must be bicubic ('degree' = 3).

'n_u_segments' The number of segments in the surface in the u direction. This must be greater than or equal to 1. Adjacent segments must meet.

'n_v_segments' The number of segments in the surface in the v direction. This must be greater than or equal to 1. Adjacent segments must meet.

'dim' This field gives the number of doubles per coefficient vector in the array 'coeffs'. Allowable values are 3 and 4.

'is_rational' This field is set to true if the surface is rational. In this case the 'dim' field must be one greater than the dimensionality of the surface.
At present, if:

'is_rational' == PK_LOGICAL_false then 'dim' must be 3
'is_rational' == PK_LOGICAL_true then 'dim' must be 4.

'rep' This field defines the representation method.
See documentation of PK_piecewise_rep_t.

'coeffs' Contains ('u_degree'+1)*'n_u_segments'*('v_degree'+1)*'n_v_segments' vectors of dimension 'dim'.
If 'is_rational' == PK_LOGICAL_false, then the vectors are 3-D vectors giving the x, y and z components.
If 'is_rational' == PK_LOGICAL_true, 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 interpretation of 'coeffs' depends on the representation method chosen; this is determined by the value of the field 'rep'.

For more information about these representations see the chapter on "The Mathematical Form of B_Geometry" in the Functional Description.

PK_BSURF_sf_t

```
struct PK_BSURF_sf_s
    {
    int u_degree;                          --- The degree in u = order-1.
    int v_degree;                          --- The degree in v = order-1.
    int n_u_vertices;                      --- The number of vertices in u.
    int n_v_vertices;                      --- The number of vertices in v.
    int vertex_dim;                        --- The dimension of each vertex.
    PK_LOGICAL_t is_rational;              --- True if the surface is rational
    double *vertex;                        --- The vertices.
    PK_BSURF_form_t form;                  --- The surface shape.
    int n_u_knots;                         --- The number of knot values in u.
    int n_v_knots;                         --- The number of knot values in v.
    int *u_knot_mult;                      --- The multiplicities in u.
    int *v_knot_mult;                      --- The multiplicities in v.
    double *u_knot;                        --- The distinct knot values in u.
    double *v_knot;                        --- The distinct knot values in v.
    PK_knot_type_t u_knot_type;            --- Enum describing the u knot set.
    PK_knot_type_t v_knot_type;            --- Enum describing the v knot set.
    PK_LOGICAL_t is_u_periodic;            --- Periodic in u.
    PK_LOGICAL_t is_v_periodic;            --- Periodic in v.
    PK_LOGICAL_t is_u_closed;              --- Closed in u.
    PK_LOGICAL_t is_v_closed;              --- Closed in v.
    PK_self_intersect_t self_intersecting; --- Yes no or maybe.
    PK_convexity_t convexity;              --- Yes no or maybe.
    };
typedef struct PK_BSURF_sf_s PK_BSURF_sf_t;
```

This data structure is the standard form for a b-surface. See the documentation below for the relationship between this data structure and STEP.

Specific Errors

PK_ERROR_bad_value	unreadable value
PK_ERROR_bad_knots	illegal knot multiplicity or knot set not monotonic increasing
PK_ERROR_wrong_number_knots	knots inconsistent with vertices
PK_ERROR_bad_dimension	dimension < 1 polynomial or < 2 rational
PK_ERROR_weight_le_0	one of the weights is zero or negative

Converting between PK_BSURF_sf_t and STEP:

This form is easily convertible to the STEP standard, but there are a number of important differences, designed to make its use simpler and more efficient.

'vertex_dim':

This field gives the number of doubles per vertex in the array 'vertex'. Allowable values are 3 and 4. A value of 3 implies a 3-d polynomial surface. A value of 4 implies a 3-d rational surface. The STEP "cartesian_point" entity allows 1, 2 or 3.

'is_rational':

This field is set to true if the surface is rational. In this case the vertex_dim field must be 4, and the weights must be included in the 'vertex' array. The STEP standard uses a distinct entity subtype "rational_b_spline_surface".

'vertex':

If the surface is polynomial, dimension 3, then the values in this field represent the vertices explicitly, in the order [x1, y1, z1, x2, y2, ...] respectively. If it is rational, dimension 4, then the cartesian points are multiplied by the weights, so that the values in this field represent x1w1, y1w1, z1w1, w1, x2w2, y2w2, ...]. The vertices are so arranged that they vary most quickly with v and then with u. The STEP standard uses a separate array of weights, and does not multiply the cartesian points by the weights.

'form':

See the documentation of 'PK_BSURF_form_t'.
This field corresponds to the field of the same name in the STEP "b_spline_surface" entity, except for the 'PK_BSURF_form_unset_c' value which does not exist in the STEP standard.

'u_knot_mult' 'v_knot_mult' 'u_knot' and 'v_knot':

The values in 'u_knot_mult' and 'v_knot_mult' give the number of times each knot is to be repeated. The minimum multiplicity allowed for any knot is 1. The maximum multiplicity allowed other than for the first and last knot is the degree in that parameter. The maximum allowed for the first or last knot is the degree+1.

The values in 'u_knot' and 'v_knot' must be distinct and form a monotone increasing set.

See the documentation below for an explanation of the required total number of knots.

These arrays correspond exactly to the fields "u_multiplicities" "v_multiplicities" "u_knots" and "v_knots" in the STEP entity sub-type "b_spline_surface_with_knots".

'u_knot_type' and 'v_knot_type':

See the documentation for 'PK_knot_type_t'. The STEP standard does not contain a field corresponding to these. The information described is conveyed by the use of entity subtypes, such as "bezier_surface". The values 'PK_knot_bezier_ends_c' and 'PK_knot_non_uniform_c' have no corresponding subtype. A surface of one of these types would be translated to STEP as a "bspline_surface_with_knots". For a surface to be translated as, for instance "uniform_surface", both knot types would have to be uniform. Otherwise it would default to the more general "bspline_surface_with_knots".

'is_u_periodic' and 'is_v_periodic':

If either of these fields is set to true, the parametrisation of the surface "wraps around". It implies that the surface is closed in that direction, with G1 continuity at the seam. This field does not exist in the STEP standard.

See the documentation below for additional requirements for the knot sets of a periodic b-surface.

'is_u_closed' and 'is_v_closed':

One or both of these fields being set to true does not necessarily imply that the parametrisation is periodic. They correspond to the fields of the same name in the STEP entity "b_spline_surface".

'self_intersecting':

    See the documentation of PK_self_intersect_t.
    This corresponds to the field "self_intersect" in the STEP entity
    "b_spline_surface", except that the STEP standard does not provide for an
    unset value.

'convexity':

    See the documentation of PK_convexity_t.
    This field has no correspondence in the STEP "b_spline_surface" entity.
    Surfaces being converted from STEP should be given a value of
    'PK_convexity_unset_c'.

The number of knots:

The knot set referred to here is the "expanded knot set" obtained by repeating each value in the array 'u_knot' or 'v_knot' the number of times given by the corresponding element in 'u_knot_mult' or 'v_knot_mult'.

A NURBS surface of degree n in one of its parameters defined over k values k-1 intervals) in that parameter requires a further n values at either end of that knot set in order for the necessary k+n-1 b-spline basis functions to be fully defined. These additional values are commonly known as the "imaginary knots". Thus the total number of knots p is related to the total number of basis functions and hence the number m of control vertices in that direction the number of rows or collumns) by the expression p = m+n+1.

Periodic Surfaces:

A periodic surface must be so constructed as to preserve G1 continuity (of tangent direction) at the seam.

In addition, it is recommended that a periodic surface be constructed with C(n-1) continuity at the seam (where n is the degree). This would give C2 continuity at the seam for a cubic. This is most easily achieved by making the "expanded knot set" "wrap around" in the sense that the intervals between the first n+1 knots (n "imaginary" knots and one "real" knot) should be the same as the corresponding intervals between the last n+1 "real" knots, and the intervals between the last n+1 knots (one "real" knot and n "imaginary" knots) should be the same as those between the first n+1 "real" knots.

The "period" of such a surface is the parameter interval separating the first and last "real" knots in the periodic parameter direction.

if indices begin at 0:

$$period = u\_or\_v_{k+n-1} - u\_or\_v_{n}$$

for i = 0 to n:

$$u\_or\_v_{k+n+i-1} - period = u\_or\_v_{n+i}$$

$$u\_or\_v_{i} + period = u\_or\_v_{k+i-1}$$

PK_BSURF_splinewise_sf_t

```
struct PK_BSURF_splinewise_sf_s
    {
    int                 u_degree;      --- The u degree = order -1
    int                 v_degree;      --- The u degree = order -1
    int                 n_u_positions; --- The number of positions in u
    int                 n_v_positions; --- The number of positions in v
    PK_VECTOR_t        *positions;     --- The positions
```

```
    PK_PARAM_end_t      u_end;          --- End condition at lower u
    PK_PARAM_end_t      v_end;          --- End condition at lower v
    PK_PARAM_end_t      U_end;          --- End condition at upper U
    PK_PARAM_end_t      V_end;          --- End condition at upper V
    PK_VECTOR_t        *u_derivs;       --- Derivatives at lower u
    PK_VECTOR_t        *v_derivs;       --- Derivatives at lower v
    PK_VECTOR_t        *U_derivs;       --- Derivatives at upper U
    PK_VECTOR_t        *V_derivs;       --- Derivatives at upper V
    PK_PARAM_knot_t     u_how_knot;     --- How u knot vector defined
    PK_PARAM_knot_t     v_how_knot;     --- How v knot vector defined
    double             *u_knot;         --- u Knot vector
    double             *v_knot;         --- v Knot vector
    PK_PARAM_twist_t    u_v;            --- how lower u lower v
                                        --- twist vector defined
    PK_PARAM_twist_t    u_V;            --- how lower u upper V
                                        --- twist vector defined
    PK_PARAM_twist_t    U_v;            --- how upper U lower v
                                        --- twist vector defined
    PK_PARAM_twist_t    U_V;            --- how upper U upper V
                                        --- twist vector defined
    PK_VECTOR_t         u_v_twist;      --- lower u lower v twist vector
    PK_VECTOR_t         u_V_twist;      --- lower u upper V twist vector
    PK_VECTOR_t         U_v_twist;      --- upper U lower v twist vector
    PK_VECTOR_t         U_V_twist;      --- upper U upper V twist vector
    };
typedef struct PK_BSURF_splinewise_sf_s PK_BSURF_splinewise_sf_t;
```

This structure is the splinewise standard form of a b-surface.

Specific Errors

```
    PK_ERROR_incompatible_props     only one end is periodic or a periodic
                                    surface has coincident end points.
    PK_ERROR_bad_derivative         a derivative is too large
    PK_ERROR_insufficient_points    not enough spline points
    PK_ERROR_coincident_points      repeated spline points
    PK_ERROR_bad_position           a spline point is outside size box
    PK_ERROR_bad_knots              bad parameterisation
    PK_ERROR_repeated_knots         repeated knots
    PK_ERROR_wrong_number_knots     wrong number of knots
```

```
'u_degree'       surface degree in the u direction.
                 When the standard form is input to
                 Parasolid (PK_BSURF_create_splinewise), 'u_degree' may only be
                 set to 3 to produce cubic curves or 0 in which case
                 Parasolid will select an appropriate degree.

'v_degree'       surface degree in the v direction.
                 When the standard form is input to
                 Parasolid (PK_BSURF_create_splinewise), 'v_degree' may only be
                 set to 3 to produce cubic curves or 0 in which case
                 Parasolid will select an appropriate degree.

'n_u_positions'  number of positions in the u direction
                 If 'u_end' is PK_PARAM_end_periodic_c then 'n_u_positions'
                 must be at least 3, otherwise 'n_u_positions' must be at
                 least 2.

'n_v_positions'  number of positions in the v direction
                 If 'v_end' is PK_PARAM_end_periodic_c then 'n_v_positions'
                 must be at least 3, otherwise 'n_v_positions' must be at
                 least 2.

'positions'      array of positions.
                 'positions' are supplied in the order with u changing most
```

```
                    rapidly.
                    Positions consecutive in either the u or v direction mus not
                    coincide.
                    For a surface which is closed but not periodic in the u
                    direction, the start and end u positions must coincide.
                    For a surface which is closed but not periodic in the v
                    direction, the start and end v positions must coincide.
                    For a surface which is periodic in the u direction, the start
                    and end u positions must not coincide.
                    For a surface which is periodic in the v direction, the start
                    and end v positions must not coincide.

'u_end'             end condition at lower u.
                    If this is PK_PARAM_end_periodic_t then 'U_end' must also be
                    PK_PARAM_end_periodic_t.
                    If this is PK_PARAM_end_clamped_vector_c then 'u_derivs'
                     must be set to the u start derivatives.

'v_end'             end condition at lower v.
                    If this is PK_PARAM_end_periodic_t then 'V_end' must also be
                    PK_PARAM_end_periodic_t.
                    If this is PK_PARAM_end_clamped_vector_c then 'v_derivs'
                    must be set to the v start derivatives.

'U_end'             end condition at upper U.
                    If this is PK_PARAM_end_periodic_t then 'u_end' must also be
                    PK_PARAM_end_periodic_t.
                    If this is PK_PARAM_end_clamped_vector_c then 'U_derivs'
                    must be set to the U end derivatives.

'V_end'             end condition at upper V.
                    If this is PK_PARAM_end_periodic_t then 'v_end' must also be
                    PK_PARAM_end_periodic_t.
                    If this is PK_PARAM_end_clamped_vector_c then 'V_derivs'
                    must be set to the V end derivatives.

'u_derivs'          derivatives at lower u if 'u_end' is
                    PK_PARAM_end_clamped_vector_c

'v_derivs'          derivatives at lower v if 'v_end' is
                    PK_PARAM_end_clamped_vector_c

'U_derivs'          derivatives at upper U if 'U_end' is
                    PK_PARAM_end_clamped_vector_c

'V_derivs'          derivatives at upper V if 'V_end' is
                    PK_PARAM_end_clamped_vector_c

'u_how_knot'        how the u knot vector is defined

'v_how_knot'        how the v knot vector is defined

'u_knot'            if 'u_how_knot' is PK_PARAM_knot_defined_c then 'u_knot' is
                    the knot vector of length 'n_u_positions' if the surface is
                    not periodic in the u direction, or 'n_u_positions' + 1 if
                    the surface is periodic in the u direction.

'v_knot'            if 'v_how_knot' is PK_PARAM_knot_defined_c then 'v_knot' is
                    the knot vector of length 'n_v_positions' if the surface is
                    not periodic in the v direction, or 'n_v_positions' + 1 if
                    the surface is periodic in the v direction.

'u_v'               how twist vector defined at lower u lower v.

'u_V'               how twist vector defined at lower u upper V.

'U_v'               how twist vector defined at upper U lower v.

'U_V'               how twist vector defined at UPPER U upper V.
```

```
'u_v_twist'      if 'u_v' is PK_PARAM_twist_yes_c the 'u_v_twist' is
                 the twist vector at lower u lower v.

'u_V_twist'      if 'u_V' is PK_PARAM_twist_yes_c the 'u_V_twist' is
                 the twist vector at lower u upper V.

'U_v_twist'      if 'U_v' is PK_PARAM_twist_yes_c the 'U_v_twist' is
                 the twist vector at upper U lower v.

'U_V_twist'      if 'U_V' is PK_PARAM_twist_yes_c the 'U_V_twist' is
                 the twist vector at upper U upper V.
```

PK_BSURF_t

This type represents a B geometry SURFace.

A bsurface is a type of surface (PK_CLASS_bsurf is a subclass of PK_CLASS_surf).

The bsurf is defined by data contained in a PK_BSURF_sf_t.
Piecewise and splinewise representations of bsurfs are also available:
See PK_BSURF_piecewise_sf_t and PK_BSURF_splinewise_sf_t

PK_check_bgeom_t

typedef int PK_check_bgeom_t;

This data type is used to select specific B-geometry checking. It has two values:

```
PK_check_bgeom_no_c             don't do B geometry validity checks
PK_check_bgeom_yes_c            check continuity of B-geometry
```

PK_check_corrupt_t

typedef int PK_check_corrupt_t;

This data type is used to select checking for corrupt data structures: It has two values:

```
PK_check_corrupt_no_c           don't do data structure corruption checks
PK_check_corrupt_yes_c          check for corrupt data structure and
                                check identifiers
```

PK_check_fa_fa_t

typedef int PK_check_fa_fa_t;

This data type is used to select checking of face-face consistency. It has two values:

```
PK_check_fa_fa_no_c           don't do face-face consistency checks
PK_check_fa_fa_yes_c          do face-face consistency checks
```

PK_check_fa_X_t

```
typedef int PK_check_fa_X_t;
```

This data type is used to select self consistency checks of faces. It has two values:

```
PK_check_fa_X_no_c          don't do face self intersection checks
PK_check_fa_X_yes_c         do face self intersection tests
```

PK_check_fault_t

```
typedef struct PK_check_fault_s
    {
    PK_check_state_t   state;      --- token indicating fault state
    PK_ENTITY_t        entity_1;   --- fault entity always present
    PK_ENTITY_t        entity_2;   --- secondary fault entity
                                   --- (may be PK_ENTITY_null)
    PK_VECTOR_t        position;   --- position within faulty volume
    } PK_check_fault_t;
```

This structure records a check fault.

Every check fault contains a 'state' and 'entity_1'. Check faults may contain an 'entity_2' and/or a 'position'. This depends upon the 'state' as shown in the following table. If 'entity_2' is not used it is set to PK_ENTITY_null, and if 'position' is not used it is set to (0.0, 0.0, 0.0).

'state'	'entity_1'	'entity_2'	'position'
PK_BODY_state_corrupt_c	body		
PK_BODY_state_invalid_ident_c	body		
PK_BODY_state_inside_out_c	body		
PK_BODY_state_bad_regions_c	body		
PK_EDGE_state_open_c	edge		
PK_EDGE_state_open_nmnl_c	edge		
PK_EDGE_state_bad_vertex_c	edge	vertex	
PK_EDGE_state_bad_vertex_nmnl_c	edge	vertex	
PK_EDGE_state_reversed_c	edge		
PK_EDGE_state_reversed_nmnl_c	edge		
PK_EDGE_state_bad_spcurve_c	edge		
PK_EDGE_state_bad_sp_nmnl_c	edge		
PK_EDGE_state_vertices_touch_c	edge		
PK_EDGE_state_bad_face_order_c	edge		
PK_EDGE_state_bad_wire_ed_ed_c	edge	edge	point in region of inconsistency
PK_ENTITY_state_invalid_c	entity		
PK_FACE_state_bad_vertex_c	face	vertex	
PK_FACE_state_bad_edge_c	face	edge	
PK_FACE_state_bad_edge_order_c	face		
PK_FACE_state_no_vtx_at_sing_c	face		point at singularity
PK_FACE_state_bad_loops_c	face		
PK_FACE_state_self_int_c	face		point in region of inconsistency
PK_FACE_state_bad_wire_fa_ed_c	face	edge	point in region of inconsistency
PK_FACE_state_check_fail_c	face	face	
PK_FACE_state_bad_face_face_c	face	face	point in region of inconsistency
PK_GEOM_state_self_int_c	geom		point on self-intersection
PK_GEOM_state_degenerate_c	geom		point in degenerate

			region
PK_REGION_state_bad_shells_c	region		
PK_SHELL_state_bad_topol_geom_c	shell		
PK_SHELL_state_bad_sh_sh_c	shell	shell	point in region of inconsistency
PK_TOPOL_state_not_G1_c	topol		
PK_TOPOL_state_size_box_c	topol		
PK_TOPOL_state_check_fail_c	topol		
PK_TOPOL_state_no_geom_c	topol		
PK_ATTDEF_state_bad_name_c	attdef		
PK_ATTRIB_state_bad_string_c	attrib		

PK_check_geom_t

typedef int PK_check_geom_t;

This data type is used to select the level of geometry checking required. It has five values:

PK_check_geom_no_c	don't do geometry validity checks
PK_check_geom_basic_c	check for invalid or degenerate geometry
PK_check_geom_lazy_c	check for invalid, degenerate or self-intersecting geometry
PK_check_geom_full_c	as for PK_check_geom_lazy_c but ignore the results of pre-V5 checks
PK_check_geom_yes_c	do all geometry checks regardless of previous checks or interface parameters

PK_check_loops_t

typedef int PK_check_loops_t;

This data type is used to select checking of loop consistency of faces. It has two values:

PK_check_loops_no_c	don't do loops checks
PK_check_loops_yes_c	check for loop concistency of faces

PK_check_sh_t

typedef int PK_check_sh_t;

This data type is used to select checking of shell consistency. It has two values:

PK_check_sh_no_c	don't do shell consistency checks
PK_check_sh_yes_c	check for inside-out or inconsistent shells

PK_check_size_box_t

typedef int PK_check_size_box_t;

This data type is used to select whether size box violation checking is required. It has two values:

PK_check_size_box_no_c	don't do size box violation checks
PK_check_size_box_yes_c	do size box violation checks

PK_check_state_t

```
typedef int PK_check_state_t;
```

PK_BODY_state_corrupt_c	data structure is corrupt
PK_BODY_state_invalid_ident_c	body has invalid or duplicate identifiers
PK_BODY_state_inside_out_c	body is inside out
PK_BODY_state_bad_regions_c	regions of body are inconsistent
PK_EDGE_state_open_c	open or non-periodic curve on ring edge
PK_EDGE_state_bad_vertex_c	vertex does not lie on edge's curve
PK_EDGE_state_reversed_c	curve in wrong direction for edge
PK_EDGE_state_bad_spcurve_c	spcurve not within edge's tolerance
PK_EDGE_state_vertices_touch_c	vertices of edge touch
PK_EDGE_state_bad_face_order_c	on a non-manifold edge, the face order around the edge does not match the order of the faces' surfaces around the edge
PK_EDGE_state_bad_wire_ed_ed_c	two wireframe edges intersect at a position other than a vertex
PK_EDGE_state_open_nmnl_c	open or non-periodic nominal geometry on ring edge
PK_EDGE_state_bad_vertex_nmnl_c	vertex does not lie on nominal geometry
PK_EDGE_state_bad_sp_nmnl_c	spcurve not within edge's tolerance of nominal geometry
PK_EDGE_state_reversed_nmnl_c	nominal geometry in wrong direction
PK_ENTITY_state_invalid_c	the entity, usually geometric, is invalid
PK_FACE_state_bad_vertex_c	vertex does not lie on face's surface
PK_FACE_state_bad_edge_c	edge does not lie on face's surface
PK_FACE_state_bad_edge_order_c	the edge order around the face does not match the order of the edges' curves around the face.
PK_FACE_state_no_vtx_at_sing_c	a surface singularity has no accompanying vertex
PK_FACE_state_bad_loops_c	loops inconsistent
PK_FACE_state_self_int_c	face is self intersecting
PK_FACE_state_bad_wire_fa_ed_c	a wireframe edge intersects a face at a position other than a vertex
PK_FACE_state_check_fail_c	checker failure during face/face check
PK_FACE_state_bad_face_face_c	face face inconsistency
PK_GEOM_state_self_int_c	self intersecting geometry
PK_GEOM_state_degenerate_c	degenerate geometry
PK_REGION_state_bad_shells_c	inconsistent shells
PK_SHELL_state_bad_topol_geom_c	a topological entity belonging to a shell is not geometrically within the shell
PK_SHELL_state_bad_sh_sh_c	shell shell inconsistency
PK_TOPOL_state_not_G1_c	topology's geometry is not G1 continuous
PK_TOPOL_state_size_box_c	some or all of entity is outside size box
PK_TOPOL_state_check_fail_c	checker failure
PK_TOPOL_state_no_geom_c	missing geometry
PK_BODY_state_ok_c	OK
PK_FACE_state_redundant_c	face redundant with respect to tolerances
PK_CURVE_state_inconsistent_c	inconsistent directions of loops of curves
PK_LOOP_state_invalid_c	invalid loop combination for surface type
PK_EDGE_state_bad_order_c	edges incorrectly ordered at vertex
PK_ATTDEF_state_bad_name_c	an attribute definition name is not transmissible in text XT format
PK_ATTRIB_state_bad_string_c	an attribute string field is not transmissible in text XT format

PK_check_top_geo_t

```
typedef int PK_check_top_geo_t;
```

This data type is used to select the level of checking of consistency between geometry and topology. It has four values:

PK_check_top_geo_no_c	don't check geometry/topology consistency
PK_check_top_geo_edge_c	just check curve/edge consistency
PK_check_top_geo_face_c	just check surface/face consistency
PK_check_top_geo_yes_c	check both curve/edge and surface/face consistency

PK_CIRCLE_array_t

This structure contains an array of entities of type PK_CIRCLE_t

```
typedef struct PK_CIRCLE_array_s
    {
    PK_CIRCLE_t    *array;   --- array of CIRCLEs
    int             length;  --- length of array
    }
    PK_CIRCLE_array_t;
```

It has the fields:

'array': array of entities of type PK_CIRCLE_t

'length': the number of entities in 'array'

PK_CIRCLE_sf_t

```
struct PK_CIRCLE_sf_s
    {
    PK_AXIS2_sf_t basis_set; --- a centre, a ref_direction and an axis
    double        radius;    --- the radius of the circle(>0)
    };

typedef struct PK_CIRCLE_sf_s PK_CIRCLE_sf_t;
```

Specific Errors

PK_ERROR_radius_le_0 radius is less than or equal to zero

The parameterisation of the circle is as follows:

```
C = basis_set.location
x = basis_set.ref_direction
y = basis_set.axis X basis_set.ref_direction ( cross product )
z = basis_set.axis
R = radius

P( t ) = C + R( ( cos t )x + ( sin t )y )  where ( 0 <= t < 2PI )
```

Radius must be larger than zero.

PK_CIRCLE_t

This type represents a complete CIRCLE.

A circle is a type of curve (PK_CLASS_circle is a subclass of PK_CLASS_curve).

The circle is defined by data contained in a PK_CIRCLE_sf_t

PK_comparison_t

```
typedef int PK_comparison_t;
```

This datatype allows the user to supply an interval and to test when given a number (d) whether the constraint is true or false. It has the following values:

```
PK_comparison_always_c       true
PK_comparison_less_c         d <= constraint.interval.value[0]
PK_comparison_equal_c        d = constraint.interval.value[0]
PK_comparison_greater_c      d >= constraint.interval.value[1]
PK_comparison_between_c      d >= constraint.interval.value[0] &&
                             d <= constraint.interval.value[1]
PK_comparison_outside_c      d <= constraint.interval.value[0] ||
                             d >= constraint.interval.value[1]
PK_comparison_never_c        false
```

PK_CONE_array_t

This structure contains an array of entities of type PK_CONE_t

```
typedef struct PK_CONE_array_s
    {
    PK_CONE_t      *array;   --- array of CONEs
    int             length;  --- length of array
    }
    PK_CONE_array_t;
```

It has the fields:

'array': array of entities of type PK_CONE_t

'length': the number of entities in 'array'

PK_CONE_sf_t

```
struct PK_CONE_sf_s
    {
    PK_AXIS2_sf_t   basis_set;  --- a point, reference direction and an axis
    double          radius;     --- the radius of a circle on the cone(>=0)
    double          semi_angle; --- the half angle of the cone in radians
    };

typedef struct PK_CONE_sf_s PK_CONE_sf_t;
```

Specific Errors

PK_ERROR_radius_lt_0	The radius for the cone was less than zero
PK_ERROR_bad_angle	The semi-angle was not strictly between 0 and PI/2

The half cone represented by this data structure passes through the circle with centre 'basis_set.location', axis 'basis_set.axis' and radius as 'radius'. The direction of the axis will be the direction of increasing circle radius (i.e pointing away from the apex in the infinite direction of the cone). The semi_angle defines the angle between the axis and any line on the surface of the cone. This is slightly different to the handling of cones in the KI.

See documentation for PK_AXIS2_sf_t.

The parameterisation of a cone is as follows:

```
P = basis_set.location
x = basis_set.ref_direction
y = basis_set.axis X basis_set.ref_direction  ( cross product )
z = basis_set.axis
R = radius
w = semi_angle

S(u, v) = P + ( R + v(tan w) )((cos u)x + (sin u)y) + vz
                              ( 0 <= u < 2PI , -r(cot w) <= v < infinity )
```

Where as in the STEP definition of a cone, the v parameter is in the range minus infinity to plus infinity, hence including two half cones sharing a common apex, only one of the two half cones is represented by this data structure.

PK_CONE_t

This type represents a semi-infinite CONE.

A cone is a type of surface (PK_CLASS_cone is a subclass of PK_CLASS_surf).

The cone is defined by data contained in a PK_CONE_sf_t.

PK_convexity_t

typedef int PK_convexity_t;

This field can take any one of the following values.

```
    'PK_convexity_unset_c'     : convexity not known
    'PK_convexity_arbitrary_c' : any convexity
    'PK_convexity_convex_c'    : convex
    'PK_convexity_concave_c'   : concave
```

PK_CURVE_approx_t

typedef int PK_CURVE_approx_t;

Curve approximation type.

PK_CURVE_approx_arclength_c	Approximating curve has arc length parameterisation.

```
    PK_CURVE_approx_even_c          Approximating curve has even
                                    parameterisation.
```

Even parametrisation means that arc length to parameter range ratio is constant along the curve.

Arc length parametrisation ensures that arc length to parameter range ratio is 1.

PK_CURVE_array_t

This structure contains an array of entities of type PK_CURVE_t

```
typedef struct PK_CURVE_array_s
    {
    PK_CURVE_t     *array;   --- array of CURVEs
    int             length;  --- length of array
    }
    PK_CURVE_array_t;
```

It has the fields:

'array': array of entities of type PK_CURVE_t

'length': the number of entities in 'array'

PK_CURVE_intersect_curve_o_t

```
struct PK_CURVE_intersect_curve_o_s
    {
    int             o_t_version;    --- version number of option structure
    PK_LOGICAL_t    have_box;       --- whether box provided (PK_LOGICAL_false)
    PK_BOX_t        box;            --- box of interest
    PK_SURF_t       common_surf;    --- surf containing curves or
                                    --- PK_ENTITY_null (PK_ENTITY_null)
    };

typedef struct PK_CURVE_intersect_curve_o_s PK_CURVE_intersect_curve_o_t;
```

Holds optional controls on intersections between two curves.

The option structure defines options applicable to the intersection of two curves.

PK_CURVE_make_approx_o_t

```
struct PK_CURVE_make_approx_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_CURVE_approx_t   approx_type;    --- approximation type
                                        --- ( PK_CURVE_approx_arclength_c )
    double              tolerance;      --- tolerance ( 0.00001 )
    };

typedef struct PK_CURVE_make_approx_o_s PK_CURVE_make_approx_o_t;
```

This options structure holds optional controls for PK_CURVE_make_approx.

Description of fields:

```
'o_t_version'           Version number of option structure.

'approx_type'           Curve approximation type. Permitted
                        values are:
                            PK_CURVE_approx_arclength_c
                            PK_CURVE_approx_even_c.
                        The default is
                        PK_CURVE_approx_arclength_c.

'tolerance'             Tolerance. This is new curve fit tolerance.
                        Default is 0.00001.
```

PK_CURVE_make_wire_body_o_t

```
struct PK_CURVE_make_wire_body_o_s
    {
    int             o_t_version;    --- version number of option structure
    double          tolerance;      --- max separation between any two curve
                                    --- endings that are to be connected
                                    --- (0.000001)
    PK_LOGICAL_t    allow_disjoint; --- is the wire body allowed to be
                                    --- disjoint? (PK_LOGICAL_true)
    PK_LOGICAL_t    check;          --- check the body for errors?
                                    --- (PK_LOGICAL_true)
    PK_LOGICAL_t    want_edges;     --- return the new edges in new_edges?
                                    --- (PK_LOGICAL_false)
    PK_LOGICAL_t    want_indices;   --- return the edge indices in edge_index?
                                    --- (PK_LOGICAL_false)
    };

typedef struct PK_CURVE_make_wire_body_o_s PK_CURVE_make_wire_body_o_t;
```

This options structure holds optional controls for PK_CURVE_make_wire_body_2.

Description of fields:

```
'o_t_version'           Version number of option structure.

'tolerance'             The maximum separation between any two curve endings
                        that are to be connected. If more than two curve ends
                        are within this distance, the two closest will be
                        joined. Curve ends seperated by a distance greater than
                        tolerance will never be connected.
                        The default is 0.000001.

'allow_disjoint'        If PK_LOGICAL_true, then disjoint bodies can be
                        created. If PK_LOGICAL_false, and the supplied set of
                        curves and bounds would result in a disjoint body, then
                        the function will fail with PK_ERROR_not_connected.
                        The default is PK_LOGICAL_true.

'check'                 The created body will be checked for errors.
                        The default is PK_LOGICAL_true.

'want_edges'            If PK_LOGICAL_true, the edges will be returned in
                        new_edge.
                        The default is PK_LOGICAL_false.

'want_indices'          If PK_LOGICAL_true, then edge_index will be filled
                        with an array of integers, indicating the position in
                        the array of curves and bounds that refers to the curve
                        and bounds used to create the corresponding edge in
                        new_edges.
                        The default is PK_LOGICAL_false.
```

PK_CURVE_spin_o_t

```
struct PK_CURVE_spin_o_s
    {
    int                     o_t_version;
    PK_LOGICAL_t            simplify;           --- (PK_LOGICAL_false)
    PK_LOGICAL_t            confine;            --- (PK_LOGICAL_false)
    PK_INTERVAL_t           interval;           --- ((0,0))
    };

typedef struct PK_CURVE_spin_o_s PK_CURVE_spin_o_t;
```

Holds optional controls on spinning a curve around an axis.

The structure has the following user fields:

'simplify' If this is set to PK_LOGICAL_false then PK_CURVE_spin_2 will always return a surface of class PK_CLASS_spun.

If its value is PK_LOGICAL_true then, if possible, PK_CURVE_spin_2 will return a 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 the curve being spun. This can occur when the whole of the received curve would sweep the spun surface twice, i.e. the surface would be self intersecting at all its points. For example, this occurs when the curve to be spun 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 the curve specified using the option field 'interval' (although the parameterisation of the new profile will, in general, be different to that of the received curve).

'confine' Where the whole of the curve supplied is to be spun, this field should be set to PK_LOGICAL_false.

If set to PK_LOGICAL_true this indicates that only a subset of the curve supplied is to be spun. Which subset of the curve is indicated by the interval specified in the option field 'interval'.

'interval' This field is used only where 'confine' is set to PK_LOGICAL_true. The interval specifies the parameter range of the part of the curve that is 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 spun. If the interval supplied specifies a section of curve that crosses the axis and PK_SESSION_set_check_self_int has been called with value PK_LOGICAL_true, then an error is signalled, otherwise (PK_SESSION_set_check_self_int was called with PK_LOGICAL_false) the part of the curve used is undefined.

If an interval is specified for cases where the curve
does not intersect the axis, the interval is ignored and the
whole curve is spun.

PK_CURVE_t

This type represents a CURVE.

A curve is a type of geometry (PK_CLASS_curve is a subclass of PK_CLASS_geom).

A curve is of one of the types:

- PK_BCURVE_t: a B geometry curve
- PK_LINE_t: an unbounded straight line
- PK_CIRCLE_t: a complete circle
- PK_ELLIPSE_t: a complete ellipse
- PK_SPCURVE_t: a Surface Parameter curve
- PK_FCURVE: a Foreign curve
- PK_ICURVE_t: an intersection curve

PK_CYL_array_t

This structure contains an array of entities of type PK_CYL_t

```
typedef struct PK_CYL_array_s
    {
    PK_CYL_t       *array;   --- array of CYLinders
    int             length;  --- length of array
    }
    PK_CYL_array_t;
```

It has the fields:

'array': array of entities of type PK_CYL_t

'length': the number of entities in 'array'

PK_CYL_sf_t

```
struct PK_CYL_sf_s
    {
    PK_AXIS2_sf_t   basis_set;  --- a point, reference direction and an axis
    double          radius;     --- the radius of the cylinder
    };

typedef struct PK_CYL_sf_s PK_CYL_sf_t;
```

Specific Errors

PK_ERROR_radius_le_0	The radius for the cylinder was less than or equal to zero

See documentation for PK_AXIS2_sf_t.

The parameterisation of a cylinder is as follows:

```
P = basis_set.location
x = basis_set.ref_direction
```

```
y = basis_set.axis X basis_set.ref_direction ( cross product )
z = basis_set.axis
R = radius(>0)

S(u, v) = P + R( (cos u)x + (sin u)y ) + vz    ( 0 <= u < 2PI )
```

PK_CYL_t

This type represents an infinite CYLinder.

A cylinder is a type of surface (PK_CLASS_cyl is a subclass of PK_CLASS_surf).

The cylinder is defined by data contained in a PK_CYL_sf_t.

PK_DEBUG_BODY_compare_o_t

```
struct PK_DEBUG_BODY_compare_o_s
    {
    int                         o_t_version;       --- version number
                                                   --- of option structure
    int                         max_diffs;         --- maximum no. face pairs
                                                   --- returned (0)
    PK_LOGICAL_t                all_tests;         --- perform all local tests
                                                   --- (PK_LOGICAL_true)
    PK_LOGICAL_t                num_tests;         --- perform numerical tests
                                                   --- (PK_LOGICAL_true)
    PK_LOGICAL_t                acc_dev_tests;     --- perform accurate
                                                   --- deviation tests
                                                   --- (PK_LOGICAL_true)
    PK_LOGICAL_t                tol_dev_tests;     --- perform tolerant
                                                   --- deviation tests
                                                   --- (PK_LOGICAL_false)
    PK_LOGICAL_t                tol_tests;         --- perform tolerance tests
                                                   --- (PK_LOGICAL_true)
    PK_LOGICAL_t                class_tests;       --- perform class tests
                                                   --- (PK_LOGICAL_true)
    PK_LOGICAL_t                sense_tests;       --- perform sense tests
                                                   --- (PK_LOGICAL_true)
    PK_LOGICAL_t                missing_tests;     --- perform missing entity
                                                   --- tests
                                                   --- (PK_LOGICAL_true)
    PK_LOGICAL_t                non_match_tests;   --- perform non-matched
                                                   --- pairs tests
                                                   --- (PK_LOGICAL_true)
    };

typedef struct PK_DEBUG_BODY_compare_o_s PK_DEBUG_BODY_compare_o_t;
```

This option structure contains the optional controls for PK_DEBUG_BODY_compare

Description of fields:

```
'max_diffs'         If 'max_diffs' = 0, only global comparison tests will
                    be performed on the bodies, for instance, comparison of
                    the number of vertices/edges or faces in the body.
                    If 'max_diffs' > 0, then more expensive tests will
                    first attempt to match face pairs, each pair consisting
                    of one face selected from each body on the basis of
                    broad similarities. Analysis will then be performed to
                    detect any differences between each face of a matched pair.
                    Analysis of remaining matched pairs will continue until
                    'max_diffs' matched face pairs have been found which are
```

	considered non-identical by those comparison tests which have been specified. The faces which remain unmatched represent a significant difference between the bodies, therefore the option to return non-matched faces is recommended. The default value of the field is zero.
'all_tests'	Perform all local comparison tests. If 'max_diffs' = 0, this argument will be ignored.
'num_tests'	Perform 'max_diffs' numerical comparison tests.
'acc_dev_tests'	Perform 'max_diffs' accurate deviation comparison tests.
'tol_dev_tests'	Perform 'max_diffs' deviation comparison tests to within session precision.
'tol_tests'	Perform 'max_diffs' tolerance comparison tests.
'class_tests'	Perform 'max_diffs' class comparison tests.
'sense_tests'	Perform 'max_diffs' sense comparison tests.
'missing_tests'	Perform 'max_diffs' missing entity comparison tests.
'non_match_tests'	Perform 'max_diffs' non-matched entity pairs comparison tests.

PK_DEBUG_BODY_compare_r_t

```
struct PK_DEBUG_BODY_compare_r_s
    {
    PK_DEBUG_global_res_t           global_result;
    int                             n_global_diffs;
    PK_DEBUG_global_diffs_r_t      *global_diffs;
    PK_DEBUG_local_res_t            local_result;
    int                             n_face_pairs;
    PK_DEBUG_face_pair_r_t         *face_pairs;
    };

typedef struct PK_DEBUG_BODY_compare_r_s PK_DEBUG_BODY_compare_r_t;
```

This structure contains the results of the body comparison.

Description of fields:

'global_result'	Whether the global comparison tests detected a difference.
'n_global_diffs'	Number of global difference structures returned
'global_diffs'	Array of global difference structures
'local_result'	Whether the specified local comparison tests detected a difference.
'n_face_pairs'	Number of matched face pairs found <= max_diffs
'face_pairs'	Array of face difference structures containing matched face pairs and the differences found between them.

PK_DEBUG_data_t

```
struct PK_DEBUG_data_s
    {
    PK_LOGICAL_t            is_geom_array;   --- whether geom_array has data
    PK_GEOM_array_t         geom_array;      --- array of geoms
    PK_LOGICAL_t            is_trim_data;    --- whether trim_data has data
    PK_SURF_trim_data_t     trim_data;       --- trim data information
    };

typedef struct PK_DEBUG_data_s PK_DEBUG_data_t;
```

This structure holds any additional data which is to be transmitted/received.

Description of fields:

'is_geom_array'	Logical indicating whether geom_array contains data.
'geom_array'	Array of geometry.
'is_trim_data'	Logical indicating whether trim data structure contains data.
'trim_data'	Trim data information.

PK_DEBUG_diff_t

```
typedef int PK_DEBUG_diff_t;
```

This data type specifies the types of differences that have been found when comparing two bodies.

PK_DEBUG_diff_n_shells_c	Different no. shells found
PK_DEBUG_diff_n_faces_c	Different no. faces found
PK_DEBUG_diff_n_loops_c	Different no. loops found
PK_DEBUG_diff_n_acc_edges_c	Different no. accurate edges found
PK_DEBUG_diff_n_tol_edges_c	Different no. tolerant edges found
PK_DEBUG_diff_n_fins_c	Different no. fins found
PK_DEBUG_diff_n_acc_vxs_c	Different no. accurate vertices found
PK_DEBUG_diff_n_tol_vxs_c	Different no. tolerant vertices found
PK_DEBUG_diff_n_vxs_c	Different no. total vertices found
PK_DEBUG_diff_n_cht_pts_c	Different no. chart points found
PK_DEBUG_diff_surf_dev_c	Matching surfaces deviate by given distance
PK_DEBUG_diff_curve_dev_c	Matching curves deviate by given distance
PK_DEBUG_diff_vx_dev_c	Matching vertices deviate by given distance
PK_DEBUG_diff_surf_class_c	Matching surfaces are of different class
PK_DEBUG_diff_curve_class_c	Matching curves are of different class
PK_DEBUG_diff_edge_tol_c	Matching edges have different tolerances
PK_DEBUG_diff_vx_tol_c	Matching vertices have different tolerances
PK_DEBUG_diff_face_sense_c	Matching faces have different sense
PK_DEBUG_diff_surf_sense_c	Matching surfaces have different sense
PK_DEBUG_diff_curve_sense_c	Matching curves have different sense
PK_DEBUG_diff_vx_missing_c	Missing vertex
PK_DEBUG_diff_face_match_c	Non-matched face found
PK_DEBUG_diff_fin_match_c	Non-matched fin found
PK_DEBUG_diff_vx_match_c	Non-matched vertex found

PK_DEBUG_face_pair_r_t

```
struct PK_DEBUG_face_pair_r_s
    {
    PK_FACE_t                       master_face;
    PK_FACE_t                       similar_face;
    int                             n_local_diffs;
    PK_DEBUG_local_diffs_r_t       *local_diffs;
    };

typedef struct PK_DEBUG_face_pair_r_s PK_DEBUG_face_pair_r_t;
```

This data structure contains any differences found between the master and similar matched face pairs (up to 10 differences per face pair).

Description of fields:

'master_face'	Matched face from master body.
'similar_face'	Matched face from similar body.
'n_local_diffs'	No. local differences found for this face pair (to a maximum of 10).
'local_diffs'	Array of return structures containing local differences.

PK_DEBUG_global_diffs_r_t

```
struct PK_DEBUG_global_diffs_r_s
    {
    PK_DEBUG_diff_t          diff;
    int                      n_masters;
    int                      n_similars;
    };

typedef struct PK_DEBUG_global_diffs_r_s PK_DEBUG_global_diffs_r_t;
```

This structure defines each global numerical difference returned from the function, detailing the entity types and the numbers of these entities in each body.

Description of fields:

'diff'	Difference type detailing type of entity returned, this can be one of shells, faces, accurate edges, tolerant edges, accurate vertices or tolerant vertices.
'n_masters'	Number of entities of the type defined in 'diff' contained in the master body.
'n_similars'	Number of entities of the type defined in 'diff' contained in the similar body.

PK_DEBUG_global_res_t

```
typedef int PK_DEBUG_global_res_t;
```

This data type specifies whether any global differences have been found when comparing two bodies.

```
    PK_DEBUG_global_res_diffs_c         global differences found
    PK_DEBUG_global_res_no_diffs_c      no global differences
```

PK_DEBUG_local_diffs_r_t

```
struct PK_DEBUG_local_diffs_r_s
    {
    PK_DEBUG_diff_t                 diff;
    PK_ENTITY_t                     master_entity;
    int                             master_int;
    double                          master_double;
    PK_CLASS_t                      master_class;
    PK_LOGICAL_t                    master_logical;
    PK_ENTITY_t                     similar_entity;
    int                             similar_int;
    double                          similar_double;
    PK_CLASS_t                      similar_class;
    PK_LOGICAL_t                    similar_logical;
    };

typedef struct PK_DEBUG_local_diffs_r_s PK_DEBUG_local_diffs_r_t;
```

This structure defining each local difference returned from the function, detailing the difference types and the difference data, where relevent, in each body.

Description of fields:

'diff'	Type of local difference returned.
'master_entity'	The entity (or PK_ENTITY_null) in the master body.
'master_int'	Field containing any integer data relating to the difference in the master body. (see table below for details)
'master_double'	Field containing any double data relating to the difference in the master body. (see table below for details)
'master_class'	Field containing any class data relating to the difference in the master body. (see table below for details)
'master_logical'	Field containing any logical data relating to the difference in the master body. (see table below for details)
'similar_entity'	The corresponding entity (or PK_ENTITY_null) in the similar body.
'similar_int'	Field containing any integer data relating to the corresponding difference in the similar body. (see table below for details)
'similar_double'	Field containing any double data relating to the corresponding difference in the similar body. (see table below for details)
'similar_class'	Field containing any class data relating to the corresponding difference in the similar body. (see table below for details)
'similar_logical'	Field containing any logical data relating to the corresponding difference in the similar body. (see table below for details)

Table of data:

Difference_type	Master Entity	Similar Entity	Description	Data returned
PK_DEBUG_diff_n_loops_c	Face	Face	Different no. loops	master_int similar int
PK_DEBUG_diff_n_fins_c	Face	Face	Different no. fins	master_int similar int
PK_DEBUG_diff_n_vxs_c	Face	Face	Different no. vertices	master_int similar_int
PK_DEBUG_diff_n_cht_pts_c	Curve	Curve	Different no. chart points on intersection curves	master_int similar_int
PK_DEBUG_diff_surf_dev_c	Surface	Surface	Non-duplicate surfaces maximum deviation distance	master_double
PK_DEBUG_diff_vx_dev_c	Vertex	Vertex	Maximum deviation distance	master_double
PK_DEBUG_diff_curve_dev_c	Curve	Curve	Non-duplicate curves maximum deviation distance	master_double
PK_DEBUG_diff_surf_class_c	Surface	Surface	Different surface classes	master_class similar class
PK_DEBUG_diff_curve_class_c	Curve	Curve	Different fin curve class	master_class similar_class
PK_DEBUG_diff_edge_tol_c	Edge	Edge	Different edge tolerance	master_double similar_double
PK_DEBUG_diff_vx_tol_c	Vertex	Vertex	Different vertex tolerance	master_double similar_double
PK_DEBUG_diff_face_sense_c	Face	Face	Different face senses	master_logical similar_logical
PK_DEBUG_diff_surf_sense_c	Surface	Surface	Different surface senses	master_logical similar_logical
PK_DEBUG_diff_curve_sense_c	Curve	Curve	Different curve sense	master_logical similar_logical
PK_DEBUG_diff_vx_missing_c	Vertex or PK_ENTITY_null		Missing vertex	
PK_DEBUG_diff_face_match_c	Face or PK_ENTITY_null		Non-matched faces	
PK_DEBUG_diff_fin_match_c	Face or PK_ENTITY_null		Non-matched fin	
PK_DEBUG_diff_vx_match_c	Vertex or PK_ENTITY_null		Non-matched isolated vertex	

NB: Deviation tests designed to detect differences in surfaces or curves, deploy modest, non-exhaustive sampling methods and are therefore unsuitable

for reliably detecting highly localised deviations in otherwise identical geometries.

PK_DEBUG_local_res_t

typedef int PK_DEBUG_local_res_t;

This data type specifies whether any local differences have been found when comparing two bodies.

```
PK_DEBUG_local_res_diffs_c         local differences found
PK_DEBUG_local_res_no_diffs_c      no local differences
```

PK_DEBUG_report_start_o_t

```
struct PK_DEBUG_report_start_o_s
    {
    int             o_t_version;        --- options structure version number
    PK_LOGICAL_t    transmit_parts;     --- whether to transmit parts
                                        --- ( PK_LOGICAL_true )
    PK_LOGICAL_t    output_as_journal;  --- whether to output the report
                                        --- embedded into a journal file
                                        --- ( PK_LOGICAL_false )
    };
typedef struct PK_DEBUG_report_start_o_s PK_DEBUG_report_start_o_t;
```

This structure provides options for the output of debug report information.

Description of fields:

'transmit_parts' — If PK_LOGICAL_true, then any necessary parts will be saved to the report file, otherwise no parts will be saved.

'output_as_journal' — If PK_LOGICAL_true, then the debug report information will be embedded into a new journal file, otherwise it will be output as a new debug report file.

PK_DEBUG_SESSION_create_cb_t

typedef void (*PK_DEBUG_SESSION_create_cb_t)(PK_ITEM_t, PK_CLASS_t);

These callback functions are registered via the PK_DEBUG_SESSION_watch functions. The create callback function is invoked whenever a watched item is created.

The callback is invoked once the item has been initialised. No modelling operation should be performed on the item.

PK_DEBUG_SESSION_destroy_cb_t

typedef void (*PK_DEBUG_SESSION_destroy_cb_t)(PK_ITEM_t, PK_CLASS_t);

These callback functions are registered via the PK_DEBUG_SESSION_watch functions. The destroy callback function is invoked whenever a watched item is deleted.

The callback is invoked just before the item is destroyed. No guarantees can be made about the state of the item and hence no modelling and/or enquiry operation should be performed on or with the item.

PK_DELTA_frustrum_t

```
typedef PK_ERROR_code_t (*PK_DELTA_open_for_write_f_t)(PK_PMARK_t,
                                                       PK_DELTA_t *);
typedef PK_ERROR_code_t (*PK_DELTA_open_for_read_f_t)(PK_DELTA_t);
typedef PK_ERROR_code_t (*PK_DELTA_close_f_t)(PK_DELTA_t);
typedef PK_ERROR_code_t (*PK_DELTA_write_f_t)(PK_DELTA_t, unsigned int,
                                              const char*);
typedef PK_ERROR_code_t (*PK_DELTA_read_f_t)(PK_DELTA_t, unsigned int, char*);
typedef PK_ERROR_code_t (*PK_DELTA_delete_f_t)(PK_DELTA_t);

struct PK_DELTA_frustrum_s
    {
    PK_DELTA_open_for_write_f_t open_for_write_fn;
                                --- open a rollmark for writing
    PK_DELTA_open_for_read_f_t open_for_read_fn;
                                --- open a rollmark for reading
    PK_DELTA_close_f_t  close_fn;    --- close a rollmark
    PK_DELTA_write_f_t  write_fn;    --- write data to a rollmark
    PK_DELTA_read_f_t   read_fn;     --- read data from a rollmark
    PK_DELTA_delete_f_t delete_fn;   --- delete a rollmark
    };
typedef struct PK_DELTA_frustrum_s PK_DELTA_frustrum_t;
```

This data structure holds pointers to six rollmark functions that the application wishes to register with the PK using PK_DELTA_register_callbacks.

PK_detail_t

```
typedef int PK_detail_t;
```

This datatype is used to classify connected facesets as details of a particular type. It has the values:

PK_detail_any_c	unspecified detail
PK_detail_rubber_c	one or more connected rubber faces
PK_detail_hole_cyl_c	any cylindrical hole
PK_detail_hole_cyl_through_c	a cylindrical hole both of whose ends are open
PK_detail_hole_cyl_blind_c	a cylindrical hole with one open end and one closed end
PK_detail_hole_cyl_closed_c	a cylindrical hole with both ends closed (ie a cavity)
PK_detail_blend_rb_const_r_c	a rolling ball blend with constant radius

Cylindrical holes must include at least one face with two loops of type PK_LOOP_type_winding_c, whose surface is of class PK_CLASS_cyl. Cylindrical holes may be closed by a face whose surface is any ruled surface (in particular a plane or cone). Cylindrical holes may include other faces bounded by two loops whose surfaces are ruled surfaces. Thus countersinks and counterbores can be included in a cylindrical hole.

PK_double_constraint_t

```
struct PK_double_constraint_s
    {
    PK_comparison_t comparison;
    PK_INTERVAL_t   interval;
    };
typedef struct PK_double_constraint_s PK_double_constraint_t;
```

This structure contains a PK_comparison_t and an interval to compare against.

PK_EDGE_array_t

This structure contains an array of entities of type PK_EDGE_t

```
typedef struct PK_EDGE_array_s
    {
    PK_EDGE_t      *array;   --- array of EDGEs
    int             length;  --- length of array
    }
    PK_EDGE_array_t;
```

It has the fields:

'array': array of entities of type PK_EDGE_t

'length': the number of entities in 'array'

PK_EDGE_ask_type_t

```
struct PK_EDGE_types_s
    {
    PK_EDGE_vertex_type_t   vertex_type; --- types of vertices on edge
    PK_enclosure_t          enclosure;   --- types of regions surrounding edge
    PK_EDGE_fins_type_t     fins_type;   --- number of fins on edge
    };

typedef struct PK_EDGE_types_s PK_EDGE_ask_type_t;
```

This structure contains classification data for an edge.

This structure contains data for a given edge that classifies it according to whether it is:

(a) open, closed, or a ring.
(b) surrounded by solid regions, by void regions, or by both.
(c) wireframe, laminar, normal, or general.

vertex_type — Whether the edge is open, closed, or a ring.
Values can be:
- PK_EDGE_type_open_c
- PK_EDGE_type_closed_c
- PK_EDGE_type_ring_c

enclosure — What types of regions the edge is surrounded by.
Values can be:
- PK_enclosure_inside_c
- PK_enclosure_outside_c
- PK_enclosure_on_c

fins_type — The type of the edge.
Values can be:

```
PK_EDGE_type_wireframe_c
PK_EDGE_type_laminar_c
PK_EDGE_type_normal_c
PK_EDGE_type_general_c
```

PK_EDGE_attach_curve_nmnl_o_t

```
struct PK_EDGE_attach_curve_nmnl_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_interval;  --- t_int is set (PK_LOGICAL_false)
    PK_INTERVAL_t       t_int;          --- Parametric interval of edge (0,1)
    };

typedef struct PK_EDGE_attach_curve_nmnl_o_s PK_EDGE_attach_curve_nmnl_o_t;
```

This options structure holds optional controls for PK_EDGE_attach_curve_nmnl.

Description of fields:

'have_interval' — Indicates whether or not a parametric interval is being passed in through the 't_int' parameter.

't_int' — If 'have_interval' is set to PK_LOGICAL_true, 't_int' should contain the parametric interval of the 'edge' argument to PK_EDGE_attach_curve_nmnl on the curve supplied. In this case, the points on the curve corresponding to the ends of the interval must lie within the vertex precision of the points at the edge's bounding vertices, or the call will fail with PK_ERROR_bad_interval.

PK_EDGE_check_o_t

```
struct PK_EDGE_check_o_s
    {
    int                 o_t_version;  --- version number of option structure
    int                 max_faults;   --- limit on fault numbers (5)
    PK_check_geom_t     geom;         --- bad geometry (PK_check_geom_yes_c)
    PK_check_bgeom_t    bgeom;        --- bad B geometry (PK_check_bgeom_yes_c)
    PK_check_top_geo_t  top_geo;      --- topology/geometry inconsistency
                                      --- (PK_check_top_geo_edge_c)
    PK_check_size_box_t size_box;     --- size box violation
                                      --- (PK_check_size_box_yes_c)
    };

typedef struct PK_EDGE_check_o_s PK_EDGE_check_o_t;
```

Holds optional controls for checking an edge.

The option structure defines options applicable to checking an edge.

PK_EDGE_fins_type_t

```
typedef int PK_EDGE_fins_type_t;
```

This datatype contains information about the type of an edge.

Values can be:

```
   PK_EDGE_type_wireframe_c   The edge is a wireframe, i.e. it has no fins.
   PK_EDGE_type_laminar_c     The edge has exactly one fin.
   PK_EDGE_type_normal_c      The edge has exactly two fins, and these have
                              opposite senses.
   PK_EDGE_type_general_c     The edge has some other combination of fins.
```

PK_EDGE_make_wire_body_o_t

```
struct PK_EDGE_make_wire_body_o_s
    {
    int            o_t_version;    --- version number of option structure
    PK_LOGICAL_t   allow_disjoint; --- the operation may return disjoint
                                   --- result (default PK_LOGICAL_false)
    PK_LOGICAL_t   copy_dep_geom;  --- copy curves with dependent geometry
                                   --- (default PK_LOGICAL_true)
    PK_LOGICAL_t   use_nmnl_geom;  --- use existing nominal geometry where
                                   --- present. (default PK_LOGICAL_false)
    double         tolerance;      --- curve approximation tolerance
                                   --- (default 0.00001)
    };

typedef struct PK_EDGE_make_wire_body_o_s PK_EDGE_make_wire_body_o_t;
```

This options structure holds optional controls for PK_EDGE_make_wire_body.

Description of fields:

'o_t_version' — Version number of option structure.

'allow_disjoint' — If set to PK_LOGICAL_true, allows creation of disjoint wires, otherwise PK_ERROR_disjoint may be raised.

'copy_dep_geom' — If set to PK_LOGICAL_true, curves with dependent geometry such as sp-curves, intersections, etc. will be simply copied to the new body, otherwise an approximation will be made.

'use_nmnl_geom' — Resulting wire will use copies of nominal geometry of existing edges where present.

'tolerance' — Tolerance to be used for curve approximation

By default geometry of original edges is copied into the new wire. It is important to remember that in some cases this involves copying dependent geometry as well. Copying an intersection will also copy two underlying surfaces.

If 'copy_dep_geom' is set to PK_LOGICAL_false than curves having dependent geometry will be approximated by BCURVEs to given 'tolerance'.

It is also possible to copy nominal geometry of the original edges, where present, into the new wire by setting 'use_nmnl_geom' to PK_LOGICAL_true. Nominal geometry may also be approximated depending on the value of 'copy_dep_geom'.

PK_EDGE_repair_o_t

```
struct PK_EDGE_repair_o_s
    {
    int                o_t_version;     --- version number of option structure
    PK_LOGICAL_t       ensure_on_surfs; --- ensure edges lie on surfaces
```

```
                                        --- (default PK_LOGICAL_true)
    double              max_tolerance;  --- maximum tolerance allowed for
                                        --- above option (default 1.0e-5).
    };

typedef struct PK_EDGE_repair_o_s PK_EDGE_repair_o_t;
```

This options structure holds optional controls for PK_EDGE_repair.

Description of fields:

'ensure_on_surfs'	Ensure that each edge lies on its adjacent surfaces, by setting an appropriate tolerance on the edge, if necessary.
'max_tolerance'	The maximum tolerance which may be set when the previous option is selected. The function will fail if it cannot make the edges lie on their surfaces by using a tolerance of at most 'max_tolerance'.

PK_EDGE_set_blend_chamfer_o_t

```
struct PK_EDGE_set_blend_chamfer_o_s
    {
    int                     o_t_version;  --- version number of option structure
    PK_blend_properties_t   properties;   --- properties of blend
                                          --- (PK_blend_properties_t)
    };

typedef struct PK_EDGE_set_blend_chamfer_o_s PK_EDGE_set_blend_chamfer_o_t;
```

Holds optional controls for setting a chamfer blend.

The option structure defines options applicable to setting a chamfer blend.

PK_EDGE_set_blend_constant_o_t

```
struct PK_EDGE_set_blend_constant_o_s
    {
    int                     o_t_version;  --- version number of option structure
    PK_EDGE_t               cliff_edge;   --- top of cliff (PK_ENTITY_null)
    PK_blend_properties_t   properties;   --- properties of blend
                                          --- (PK_blend_properties_t)
    };

typedef struct PK_EDGE_set_blend_constant_o_s PK_EDGE_set_blend_constant_o_t;
```

Holds optional controls for setting a constant radius blend.

The option structure defines options applicable to setting a constant radius blend.

PK_EDGE_set_blend_variable_o_t

```
struct PK_EDGE_set_blend_variable_o_s
    {
    int                     o_t_version;  --- version number of option structure
    PK_blend_properties_t   properties;   --- properties of blend
                                          --- (PK_blend_properties_t)
    };

typedef struct PK_EDGE_set_blend_variable_o_s PK_EDGE_set_blend_variable_o_t;
```

Holds optional controls for setting a variable radius blend.

The option structure defines options applicable to setting a variable radius blend.

PK_EDGE_t

This type represents a EDGE. An edge is a bounded connected subset of a curve.

An edge is a type of topology (PK_CLASS_edge is a subclass of PK_CLASS_topol).

PK_EDGE_vertex_type_t

```
typedef int PK_EDGE_vertex_type_t;
```

This datatype indicates the closure state of an edge.

Values can be:

PK_EDGE_type_open_c	The edge has two distinct vertices.
PK_EDGE_type_closed_c	The edge has the same vertex at its start and end.
PK_EDGE_type_ring_c	The edge has no vertices.

PK_ELLIPSE_array_t

This structure contains an array of entities of type PK_ELLIPSE_t

```
typedef struct PK_ELLIPSE_array_s
    {
    PK_ELLIPSE_t   *array;   --- array of ELLIPSEs
    int             length;  --- length of array
    }
    PK_ELLIPSE_array_t;
```

It has the fields:

'array': array of entities of type PK_ELLIPSE_t

'length': the number of entities in 'array'

PK_ELLIPSE_sf_t

```
struct PK_ELLIPSE_sf_s
    {
    PK_AXIS2_sf_t basis_set;    --- a centre, a major axis and an axis
    double        R1;           --- the larger radius of the ellipse(>0)
    double        R2;           --- the smaller radius of the ellipse(>0)
    };

typedef struct PK_ELLIPSE_sf_s PK_ELLIPSE_sf_t;
```

Specific Errors

```
PK_ERROR_radius_le_0            radius less than or equal to zero
PK_ERROR_R1_R2_mismatch         R1 less than or equal to R2
```

The parameterisation of the ellipse is as follows:

```
    C = basis_set.location
    x = basis_set.ref_direction                  ( major axis )
    y = basis_set.axis X basis_set.ref_direction ( cross product )
    z = basis_set.axis
    R = R1
    r = R2

    P( t ) = C + (R cos t)x + (r sin t)y

                                where  ( 0 <= t < 2PI )
```

R1 MUST be larger than R2. Both R1 and R2 must be larger than zero.

PK_ELLIPSE_t

This type represents a complete ELLIPSE.

An ellipse is a type of curve (PK_CLASS_ellipse is a subclass of PK_CLASS_curve).

The ellipse is defined by data contained in a PK_ELLIPSE_sf_t

PK_enclosure_t

```
typedef int PK_enclosure_t;

    PK_enclosure_inside_c
    PK_enclosure_outside_c
    PK_enclosure_on_c
```

PK_ENTITY_array_t

This structure contains an array of entities of type PK_ENTITY_t

```
typedef struct PK_ENTITY_array_s
    {
    PK_ENTITY_t    *array;   --- array of ENTITYs
    int             length;  --- length of array
```

```
    }
    PK_ENTITY_array_t;
```

It has the fields:

'array': array of entities of type PK_ENTITY_t

'length': the number of entities in 'array'

PK_ENTITY_ask_description_o_t

```
struct PK_ENTITY_ask_description_o_s
    {
    int                         o_t_version; --- version number
    };
typedef struct PK_ENTITY_ask_description_o_s PK_ENTITY_ask_description_o_t;
```

This option structure controls the textual description string returned by PK_ENTITY_ask_description. It currently has no effect.

PK_ENTITY_t

This type represents an ENTITY. An entity is any modelling item.

An entity may be of one of the types:

- PK_ATTDEF_t: an attribute definition
- PK_ATTRIB_t: an attribute
- PK_GROUP_t: a group
- PK_TRANSF_t: a transform
- PK_TOPOL_t: any topological entity
- PK_GEOM_t: any geometrical entity

PK_ERROR_code_t

```
typedef int PK_ERROR_code_t;
```

This datatype contains the error codes that may be returned by a call to a PK function.

For a complete list of the error codes that may be returned from a PK function call, see either:

* PK Interface Programming Reference, Appendix C, Error Codes (Numeric)
* PK Interface Programming Reference, Appendix D, Error Codes (Alphabetic)

For more information about error handling, see Ch 57, Error Handling, in the Parasolid Functional Description.

PK_ERROR_frustrum_t

```
typedef PK_ERROR_code_t (*PK_ERROR_handler_f_t)( PK_ERROR_sf_t * );

struct PK_ERROR_frustrum_s
    {
    PK_ERROR_handler_f_t handler_fn;
    };
typedef struct PK_ERROR_frustrum_s PK_ERROR_frustrum_t;
```

This data structure holds a pointer to the error handling function the application wishes to register with the PK using PK_ERROR_register_callbacks.

The handler must be a function which receives a standard error structure and returns an error_code.

PK_ERROR_severity_t

```
typedef int PK_ERROR_severity_t;
```

This datatype describes the severity of an error returned from a PK function call.

One of the following values:

```
PK_ERROR_none    No error.
PK_ERROR_mild    Mild error.
PK_ERROR_serious Serious error.
PK_ERROR_fatal   Fatal error.
```

For more information about error severity, and the appropriate action to take after each type of error, see Ch 57, Error Handling, in the Parasolid Functional Description.

PK_ERROR_sf_t

```
struct PK_ERROR_sf
    {
    char function[PK_max_token_size];      --- name of PK function in which
                                           --- error occurred
    PK_ERROR_code_t code;                  --- error code
    char code_token[PK_max_token_size];    --- name of error code
    PK_ERROR_severity_t severity;          --- mild, serious or fatal
    int argument_number;                   --- number of invalid argument or 0
    char argument_name[PK_max_token_size]; --- argument name if arg no != 0
    int argument_index;                    --- if arg no != 0 and argument
                                           --- is an array, the array index
                                           --- where the checking failed
    PK_ENTITY_t entity;                    --- entity to which error applies or
                                           --- PK_ENTITY_null
    };
typedef struct PK_ERROR_sf PK_ERROR_sf_t;
```

This data structure is the standard form for an error record used for reporting errors to application code.

This is used to pass information to the application supplied error handler and to PK_ERROR_ask_last.

PK_FABORT_f_t

```
typedef void (*PK_FABORT_f_t)(int *ifail);

  Called at the end of an aborted kernel operation

Returns:
  int                     *ifail          --- error code: FR_no_errors

Description:
  This function is called by the kernel following a kernel operation which
  was aborted by a call to PK_SESSION_abort. It allows the application to
  do any generic tidying following the abort and/or do a long-jump back to
  a "recovery-point" within the application code (this is the only case
  where it is legitimate for a frustrum function to do such a long-jump).
  See PK_SESSION_abort for further details.
```

PK_FACE_array_t

This structure contains an array of entities of type PK_FACE_t

```
typedef struct PK_FACE_array_s
    {
    PK_FACE_t      *array;   --- array of FACEs
    int             length;  --- length of array
    }
    PK_FACE_array_t;

It has the fields:

'array':   array of entities of type PK_FACE_t

'length':  the number of entities in 'array'
```

PK_FACE_boolean_o_t

```
struct PK_FACE_boolean_o_s
    {
    int                         o_t_version;            --- version number
    PK_boolean_function_t       function;               --- (PK_boolean_unite_c)
    const PK_boolean_config_o_t
                               *configuration;          --- (NULL)
    const PK_boolean_region_select_o_t
                               *select_region;          --- (NULL)
    const PK_boolean_match_o_t
                               *matched_region;         --- (NULL)
    PK_LOGICAL_t                extend_face_list;       --- (PK_LOGICAL_false)
    PK_LOGICAL_t                stop_self_intersection; --- (PK_LOGICAL_false)
    PK_LOGICAL_t                tracking;               --- (PK_LOGICAL_false)
    PK_LOGICAL_t                merge_imprinted;        --- (PK_LOGICAL_false)
    PK_boolean_fence_t          fence;            --- (PK_boolean_fence_none_c)
    PK_LOGICAL_t                allow_disjoint;         --- (PK_LOGICAL_false)
    PK_LOGICAL_t                selective_merge;        --- (PK_LOGICAL_false)
    PK_boolean_check_fa_t       check_fa;         --- (PK_boolean_check_fa_yes_c)
    double                      default_tol;            --- (1.0E-5)
    double                      max_tol;                --- (0.0)
    PK_LOGICAL_t                merge_attributes;       --- (PK_LOGICAL_false)
    PK_LOGICAL_t                keep_target_edges;      --- (PK_LOGICAL_false)
    };
typedef struct PK_FACE_boolean_o_s PK_FACE_boolean_o_t;
```

This option structure is supplied to a face boolean.

Description of fields:

'function'	Indicates whether a intersect, unite or subtract operation is to be performed, permitted values are PK_boolean_intersect_c PK_boolean_subtract_c PK_boolean_unite_c
'configuration'	Describes the configuration of target and tool bodies.
'select_region'	Used to identify regions of the tool to include or exclude in the result.
'matched_region'	Describes matched regions between the target and tool bodies.
'extend_face_list'	A face boolean receives a subset of the faces of a target body. If the imprinting phase of the boolean results in incomplete loops of imprinted edges, then additional target faces will be used.
'stop_self_intersection'	Stop self intersection, after imprinting, tool faces which will survive the boolean are then compared against all faces of the target body.
'tracking'	Request additional topological data on the result of the boolean, this is specific to the function being called.
'merge_imprinted'	Merge all mergeable imprinted edges created and/or located by the boolean operation. Note that an existing edge can be considered an imprinted edge if it forms part of the intersection between faces from the target and tool.
'fence'	To determine which resultant bodies are returned when performing solid/sheet (trimming) or sheet/solid (punching) boolean operations. Permitted values are : PK_boolean_fence_front_c PK_boolean_fence_back_c PK_boolean_fence_none_c
'allow_disjoint'	If this option is set to PK_LOGICAL_true, the result of the boolean will be a single body which may contain disjoint manifold components. This option is ignored when a solid target and sheet tool body are specified for a subtract operation and the 'fence' option is specified with a value of PK_boolean_fence_none_c.
'selective_merge'	If this is set to PK_LOGICAL_true, the merge phase will avoid mergeing imprinted edges that existed and were mergeable before the start of the boolean. This flag has no effect if 'merge_imprinted' is not set to PK_LOGICAL_true.
'check_fa'	The value of this determines if face checking is performed during the operation, permitted values are : PK_boolean_check_fa_no_c PK_boolean_check_fa_yes_c The default is set to 'PK_boolean_check_fa_yes_c' and it is strongly advised not to change this value.

If the face checking is switched off, the boolean operation may produce an invalid body without raising an error.

'default_tol' — The default tolerance to which the boolean operation, if it has to, may approximate entities or assume that entities are coincident. This value should be set to a value that is related to the tolerance to which the application is creating approximations.
If this field is set to zero (0.0), it is assumed that the decision to set tolerances on entities is being left to the modeller.

'max_tol' — This value specifies the maximum tolerance which may be applied to any entity involved in the boolean operation.
If this field is set to zero (0.0), the default, then it assumed that no upper limit has been set on the tolerance of any entity.
If the field is not set to zero (0.0), then it must be set to a value that is greater than the value of the 'default_tol' field.

'merge_attributes' — If this is set to PK_LOGICAL_true, then any situation in which two coincident faces are combined to result in a face in a result body, will be treated as a 'merge' event from the point of view of attribute processing and attribute callbacks. If it is set to PK_LOGICAL_false, which is the default, then such a situation will be treated as the deletion of one of the faces. A subtract operation will never result in merge events.

'keep_target_edges' — This option controls which edge survives when a target edge and a tool edge coincide in a boolean. If both faces adjacent to the tool edge survive, then the tool edge will also survive. Otherwise, the result depends on the setting of the option:

PK_LOGICAL_true:
The target edge survives, unless the target edge is a new edge imprinted during the boolean, and the tool edge is an original edge - in this case, the tool edge survives.

PK_LOGICAL_false (the default):
The older edge (i.e. the one with the smaller tag) survives.

This option only has an effect if general topology is disabled. If it is enabled, then the target edge will always survive, even if it is a newly imprinted edge.

PK_FACE_change_blend_o_t

```
struct PK_FACE_change_blend_o_s
    {
    int             o_t_version;    --- version number of option structure
    double          radius;         --- radius of blend when reapplied (0.0)
    };
typedef struct PK_FACE_change_blend_o_s PK_FACE_change_blend_o_t;
```

This structure contains blend-specific optional controls for PK_FACE_change.

Description of fields:

'radius' The value in 'radius' defines the blend radius to be used when reapplying the blend and must take a value greater than or equal to zero. If 'radius' is set to 0.0 (the default), the radius of the blend when reapplied will be the same as that of the original blend.

PK_FACE_change_data_blend_t

```
typedef void* PK_FACE_change_data_blend_t;
```

Parameters of change operation blend (none).

PK_FACE_change_data_offset_t

```
typedef struct
    {
    double   distance;              --- offset distance
    } PK_FACE_change_data_offset_t;
```

Parameters of change operation offset.

Description of fields:

'distance' Distance the faces will be offset by.

PK_FACE_change_data_replace_t

```
typedef struct
    {
    PK_SURF_t     surface;          --- replace surface
    PK_LOGICAL_t  sense;            --- replace surface orientation
    } PK_FACE_change_data_replace_t;
```

Parameters of change operation replace.

Description of fields:

'surface' Replacement surface for the relevant face.

'sense' Replacement surface orientation. For a particular face/surface combination, this flag is interpreted as follows :

PK_LOGICAL_true :
When the new surface is attached to the face, the face normal must be in the same direction as the natural surface normal (as calculated by the function PK_SURF_eval_with_normal).

PK_LOGICAL_false :
When the new surface is attached to the face, the face normal must be in the opposite direction to the natural surface normal (as calculated by the function PK_SURF_eval_with_normal).

PK_FACE_change_data_t

```
union PK_FACE_change_data_u
    {
    PK_FACE_change_data_offset_t    offset;    --- offset parameters
    PK_FACE_change_data_taper_t     taper;     --- taper parameters
    PK_FACE_change_data_transform_t transform; --- transform parameters
    PK_FACE_change_data_replace_t   replace;   --- replace parameters
    PK_FACE_change_data_blend_t     blend;     --- blend parameters
    };
typedef union PK_FACE_change_data_u PK_FACE_change_data_t;
```

This union groups the definition of all the different change operation parameters used in PK_FACE_change.

Description of fields:

'offset'	The operation parameter structure is interpretated as an offset parameter structure.
'taper'	The operation parameter structure is interpretated as a taper parameter structure.
'transform'	The operation parameter structure is interpretated as a transform parameter structure.
'replace'	The operation parameter structure is interpretated as a replace-surface parameter structure.
'blend'	The operation parameter structure is interpretated as a blend parameter structure.

PK_FACE_change_data_taper_t

```
typedef struct
    {
    PK_VECTOR_t         direction;       --- taper direction
    double              angle;           --- taper angle (radian)
    int                 n_refs;          --- number of reference entities
    const PK_ENTITY_t  *references;      --- taper reference entities
    } PK_FACE_change_data_taper_t;
```

Parameters of change operation taper. For more details about the signification of these parameters, please see PK_FACE_taper.

Description of fields:

'direction'	Taper direction.
'angle'	Taper angle (in radians).
'n_refs'	Size the 'references' array.
'references'	Array of the taper reference entities (cf PK_FACE_taper).

PK_FACE_change_data_transform_t

```
typedef struct
    {
    PK_TRANSF_t  transform;        --- transform
    } PK_FACE_change_data_transform_t;
```

Parameters of change operation transform.

Description of fields:

'transform' Transform to be applied to the faces.

PK_FACE_change_o_t

```
struct PK_FACE_change_o_s
    {
    int                         o_t_version;    --- version number of option
                                                --- structure
    PK_LOGICAL_t                merge_face;     --- whether changed faces are to
                                                --- be merged with their
                                                --- neighbours, if feasible
                                                --- (PK_LOGICAL_true)
    PK_LOGICAL_t                allow_disjoint; --- whether disjoint body can
                                                --- be produced
                                                --- (PK_LOGICAL_false)
    PK_check_fa_fa_t            check_fa_fa;    --- check for face-face
                                                --- inconsistencies
                                                --- (PK_check_fa_fa_yes_c)

    PK_replace_edge_data_t      edge_data;      --- replace data for edges
    PK_replace_vertex_data_t    vertex_data;    --- replace data for vertices
    PK_replace_help_points_t    edge_help;      --- edge help points
    PK_replace_help_points_t    vertex_help;    --- vertex help points
    };
typedef struct PK_FACE_change_o_s PK_FACE_change_o_t;
```

This structure contains optional controls for PK_FACE_change.

Description of fields:

'merge_face' If this flag is set to PK_LOGICAL_true, the new faces produced by the operation will be merged with their neighbouring faces, if it is feasible. The default value is PK_LOGICAL_true.

'allow_disjoint' If this flag is set to PK_LOGICAL_true, and if the resulting body is split by the operation, all the disjoint components of that body will be preserved within a single body. If this flag is set to PK_LOGICAL_false, only one disjoint part of the body will be returned.
The default value is PK_LOGICAL_false.

'check_fa_fa' Check for face-face inconsistencies. Permitted values are:

PK_check_fa_fa_yes_c
PK_check_fa_fa_no_c

The default value is PK_check_fa_fa_yes_c.

'edge_data' Structure containing the data relative to the edge curve replacements. Refer to its documentation for details.

'vertex_data'	Structure containing the data relative to the vertex point replacements. Refer to its documentation for details.
'edge_help'	Structure containing the data relative to the edge help points. Help points are used to pick new edge curves. In case of multiple solutions, the closest curve to the related help point is chosen.
'vertex_help'	Structure containing the data relative to the vertex help points. Help points are used to pick new vertex positions. In case of multiple solutions, the closest point to the related help point is chosen.

PK_FACE_change_offset_o_t

```
struct PK_FACE_change_offset_o_s
    {
    int     o_t_version;
    };
typedef struct PK_FACE_change_offset_o_s PK_FACE_change_offset_o_t;
```

There is currently no offset specific options.

PK_FACE_change_opts_t

```
union PK_FACE_change_opts_u
    {
    PK_FACE_change_offset_o_t    *offset;    --- offset specific options
    PK_FACE_change_taper_o_t     *taper;     --- taper specific options
    PK_FACE_change_transform_o_t *transform; --- transform specific options
    PK_FACE_change_replace_o_t   *replace;   --- replace specific options
    PK_FACE_change_blend_o_t     *blend;     --- blend specific options
    };
typedef union PK_FACE_change_opts_u PK_FACE_change_opts_t;
```

This union groups the different possible types of operation specific options.

Description of fields:

'offset'	An options structure containing offset-specific optional controls for PK_FACE_change
'taper'	An options structure containing taper-specific optional controls for PK_FACE_change
'transform'	An options structure containing transform-specific optional controls for PK_FACE_change
'replace'	An options structure containing replace-specific optional controls for PK_FACE_change
'blend'	An options structure containing blend-specific optional controls for PK_FACE_change

PK_FACE_change_replace_o_t

```
struct PK_FACE_change_replace_o_s
    {
    int                   o_t_version;  --- version number of option structure
    PK_replace_merge_t    merge;        --- whether changed faces are to be
                                        --- merged with their neighbours if
                                        --- feasible (PK_replace_merge_no_c)
    };
typedef struct PK_FACE_change_replace_o_s PK_FACE_change_replace_o_t;
```

This structure contains replace-surface-specific optional controls for PK_FACE_change

Description of fields:

'merge' Adjacent faces may become mergeable when the surface of one or both faces is replaced. Permitted values are:

PK_replace_merge_no_c
PK_replace_merge_in_c
PK_replace_merge_out_c

The default value is PK_replace_merge_no_c

PK_FACE_change_t

```
struct PK_FACE_change_s
    {
    PK_FACE_change_type_t  op_type;    --- change operation type
    PK_FACE_change_data_t  op_param;   --- change operation parameters
    PK_FACE_change_opts_t  op_opts;    --- change operation options
    };
typedef struct PK_FACE_change_s PK_FACE_change_t;
```

This structure contains the definition of a change operation.

Description of fields:

'op_type' Type of the change operation to perform.
Values can be :
PK_FACE_change_type_none_c
PK_FACE_change_type_offset_c
PK_FACE_change_type_taper_c
PK_FACE_change_type_transform_c
PK_FACE_change_type_replace_c
PK_FACE_change_type_blend_c
The default value is PK_FACE_change_type_none_c.

'op_param' Parameters of the change operation. This field is a union of the different possible change operation parameters (cf PK_FACE_change_data_t).

'op_opts' Change-operation-specific options. This field is a union of the different possible change operation parameters (cf PK_FACE_change_opts_t).

PK_FACE_change_taper_o_t

```
struct PK_FACE_change_taper_o_s
    {
    int               o_t_version;  --- version number of option structure
    PK_EDGE_array_t   tapered_edges;--- tapered step edges
```

```
    PK_EDGE_array_t   normal_edges; --- normal step edges
    PK_taper_method_t method;       --- main face taper method
                                    --- (PK_taper_method_curve_c)
    };
typedef struct PK_FACE_change_taper_o_s PK_FACE_change_taper_o_t;
```

This structure contains taper-specific optional controls for PK_FACE_change

Description of fields:

'tapered_edges'	Edges which will give rise to tapered step faces. These faces' surfaces will satisfy the taper condition. Edges in this array should NOT also appear in the 'normal_edges' array.
'normal_edges'	Edges which will give rise to normal step faces. These faces' surfaces are ruled surfaces normal to the relevant original taper face. These surfaces do NOT necessarily satisfy the taper condition. Edges in this array should NOT also appear in the 'tapered_edges' array.
'method'	Taper method to construct main taper faces' surfaces. Values can be : PK_taper_method_curve_c : curve method PK_taper_method_surface_c : surface method The default value is PK_taper_method_curve_c. For more details on taper surfaces refer to the Definitions of Isocline Curves and Surfaces section of the Functional Description.

PK_FACE_change_transform_o_t

```
struct PK_FACE_change_transform_o_s
    {
    int     o_t_version;
    };
typedef struct PK_FACE_change_transform_o_s PK_FACE_change_transform_o_t;
```

There is currently no transform specific options.

PK_FACE_change_type_t

```
typedef int PK_FACE_change_type_t;
```

Types of change operations.

Values can be:

```
   PK_FACE_change_type_none_c
   PK_FACE_change_type_offset_c
   PK_FACE_change_type_taper_c
   PK_FACE_change_type_transform_c
   PK_FACE_change_type_replace_c
   PK_FACE_change_type_blend_c
```

The default value is PK_FACE_change_type_none_c.

PK_FACE_check_o_t

```
struct PK_FACE_check_o_s
    {
    int                 o_t_version;  --- version number of option structure
    int                 max_faults;   --- limit on fault numbers (5)
    PK_check_geom_t     geom;         --- bad geometry (PK_check_geom_yes_c)
    PK_check_bgeom_t    bgeom;        --- bad B geometry (PK_check_bgeom_yes_c)
    PK_check_top_geo_t  top_geo;      --- topology/geometry inconsistency
                                      --- (PK_check_top_geo_yes_c)
    PK_check_size_box_t size_box;     --- size box violation
                                      --- (PK_check_size_box_yes_c)
    PK_check_fa_X_t     fa_X;         --- face self intersection
                                      --- (PK_check_fa_X_yes_c)
    PK_check_loops_t    loops;        --- loop checks (PK_check_loops_yes_c)
    };

typedef struct PK_FACE_check_o_s PK_FACE_check_o_t;
```

Holds optional controls for checking a face.

The option structure defines options applicable to checking a face.

PK_FACE_check_pair_o_t

```
struct PK_FACE_check_pair_o_s
    {
    int                 o_t_version;  --- version number of option structure
    int                 max_faults;   --- limit on fault numbers (1)
    };

typedef struct PK_FACE_check_pair_o_s PK_FACE_check_pair_o_t;
```

Holds optional controls for checking a pair of faces.

The option structure defines options applicable to checking a pair of faces.

PK_FACE_close_gaps_o_t

```
struct PK_FACE_close_gaps_o_s
    {
    int             o_t_version;
    };

typedef struct PK_FACE_close_gaps_o_s PK_FACE_close_gaps_o_t;
```

This option structure allows for potential expansion of the close_gaps functionality. It currently has no application-settable fields.

PK_FACE_coi_t

```
typedef int PK_FACE_coi_t;

PK_FACE_coi_t will be one of the following
    PK_FACE_coi_yes_c          Two faces are coincident to the given tolerance.

    PK_FACE_coi_yes_reversed_c Two faces are coincident to the given tolerance
                               but their normals are reversed.
```

PK_FACE_coi_no_topol_c	The two faces are not coincident. The numbers of holes are different. Note that isolated loops and wire edges are ignored.
PK_FACE_coi_no_bound_1_c	The two faces are not coincident. At least one point on the boundary of face No. 1 is not coincident with the boundary of face No. 2. The point is returned.
PK_FACE_coi_no_bound_2_c	The two faces are not coincident. At least one point on the boundary of face No. 2 is not coincident with the boundary of face No. 1. The point is returned.
PK_FACE_coi_no_face_1_c	The two faces are not coincident. At least one point on face No. 1 is not coincident with face No. 2. The point is returned.
PK_FACE_coi_no_face_2_c	The two faces are not coincident. At least a point on face No. 2 is not coincident with face No. 1. The point is returned.
PK_FACE_coi_no_rubber_c	At least one of the faces does not have a surface attached.

PK_FACE_contains_vectors_o_t

```
struct PK_FACE_contains_vectors_o_s
    {
    int          o_t_version;       --- version number of options structure
    int          n_uvs;             --- number of uv positions (0)
    PK_UV_t     *uvs;               --- uv positions (NULL)
    int          n_vectors;         --- number of xyz positions (0)
    PK_VECTOR_t *vectors;           --- xyz positions (NULL)
    int          n_loops;           --- number of loops (0)
    PK_LOOP_t   *loops;             --- loops (NULL)
    PK_LOGICAL_t is_on_surf;        --- whether position vector is assumed
                                    --- to be on face's surface
                                    --- (PK_LOGICAL_true)
    };
typedef struct PK_FACE_contains_vectors_o_s PK_FACE_contains_vectors_o_t;
```

Specific Errors

PK_ERROR_nitems_le_0	'n_uvs' and 'n_vectors' are both zero. There are no points to test.
PK_ERROR_not_same_length	'n_uvs' and 'n_vectors' are both non-zero but not the same

This option structure provides variable form input to PK_FACE_contains_vectors. Positions for testing may be supplied as parametric positions on the surface of the face, or as position vectors in 3D space, or both.
Where both are supplied, corresponding elements of each array must represent the same position, and the number of elements in each array must be the same.

What is used as the boundary of the face may be restricted to a given set of 'loops'. EG if just the outer loop of a face is given then any holes in the face will not be taken into account.

'is_on_surf' indicates whether 'vectors' are known to be on the face's surface.

If 'is_on_surf' is PK_LOGICAL_true then it is assumed that any given 'vectors' lie on the face's surface. If a vector is given off the surface then spurious results may be returned.

If 'is_on_surf' is PK_LOGICAL_false, then any given vector which does not lie on the face's surface will always be classified as being outside the face.

PK_FACE_delete_blends_o_t

```
struct PK_FACE_delete_blends_o_s
    {
    int              o_t_version;    --- version number of option structure
    PK_check_fa_fa_t check_fa_fa;    --- check for face-face inconsistencies
                                     --- (PK_check_fa_fa_yes_c)
    };
typedef struct PK_FACE_delete_blends_o_s PK_FACE_delete_blends_o_t;
```

This structure contains optional controls for PK_FACE_delete_blends

Description of fields:

'check_fa_fa' Check for face-face inconsistencies. Permitted values are:
PK_check_fa_fa_yes_c
PK_check_fa_fa_no_c
The default value is PK_check_fa_fa_yes_c.

PK_FACE_delete_facesets_o_t

```
struct PK_FACE_delete_facesets_o_s
    {
    int            o_t_version;    --- options structure version number
    PK_LOGICAL_t   allow_disjoint; --- allow function to produce disjoint
                                   --- bodies (PK_LOGICAL_false)
    PK_FACE_heal_t heal_action;    --- healing action (PK_FACE_heal_yes_c)
    int            n_details;      --- number of details (0)
    PK_detail_t   *details;        --- array of details (NULL)
    double         tolerance;      --- tolerance (1.0e-6)
    };
typedef struct PK_FACE_delete_facesets_o_s PK_FACE_delete_facesets_o_t;
```

This structure contains options for PK_FACE_delete_facesets

Its user fields are:

'allow_disjoint' if set to PK_LOGICAL_true, then when the operation splits the body into two or more components then the function will return a single disjoint body. If set to PK_LOGICAL_false (the default) then the components are turned into separate bodies.

'heal_action' If set to PK_FACE_heal_yes (the default), then the function will attempt to heal wounds by whatever method it can. If set to PK_FACE_heal_no, then wounds will not be healed and each faceset will be replaced by a single rubber face. If set to any other value then only the given heal_action will be attempted on each wound.

'n_details' number of details in the following array. Setting this to zero is equivalent to giving an array of length one containing PK_detail_any_c. Otherwise the array 'details' is an array parallel to the 'facesets' given to PK_FACE_delete_facesets and indicates the detail type of each faceset. If 'n_details' is less than 'n_facesets' then the final element of 'details' is repeated for the remaining facesets

'details' An array of details giving the detail type for each faceset which may be used to select appropriate healing methods.

'tolerance' tolerance for healing

RESTRICTIONS:

The 'allow_disjoint' equals PK_LOGICAL_true option is not implemented.

'heal_action' may only be set to PK_FACE_heal_yes_c or PK_FACE_heal_no_c

PK_FACE_heal_t

```
typedef int PK_FACE_heal_t;
```

PK_FACE_heal_no_c	don't heal wounds - leave rubber faces
PK_FACE_heal_cap_c	find a surface in which all edges of a hole lie and attach this to a face covering the hole
PK_FACE_heal_grow_from_child_c	faces around the hole are extended until they cover the hole
PK_FACE_heal_grow_from_parent_c	grow the faces the parent had around the hole until they cover it
PK_FACE_heal_shrink_c	if extending faces does not yield a solution, then shrinking the faces is tried
PK_FACE_heal_yes_c	attempt healing by any appropriate means

PK_FACE_hollow_o_t

```
struct PK_FACE_hollow_o_s
    {
    int                o_t_version;     --- version number of option structure
    PK_check_fa_fa_t   check_fa_fa;     --- check for face-face
                                        --- (PK_check_fa_fa_yes_c)
    PK_LOGICAL_t       pierce_tangent;  --- process tangent pierce faces before
                                        --- offset operation
                                        --- (PK_LOGICAL_false)
    PK_offset_method_t offset_method;   --- offset method
                                        --- (PK_offset_method_accurate_c)
    };
typedef struct PK_FACE_hollow_o_s PK_FACE_hollow_o_t;
```

This structure contains optional controls for PK_FACE_hollow_3

Description of fields:

'check_fa_fa' Check for face-face inconsistencies. Permitted values are:

PK_check_fa_fa_yes_c
PK_check_fa_fa_no_c

The default value is PK_check_fa_fa_yes_c.

'pierce_tangent' A "tangent pierce face" is a pierce face which shares at least one smooth edge with a face which is not a pierce face. When this option is set to PK_LOGICAL_true, any tangent pierce face(s) will be processed before offsetting the faces in the body so that side faces can be created along the smooth boundary edges. Setting this option to PK_LOGICAL_true has no effect on the result if the body has no tangent pierce faces.

```
'offset_method'         The method used for offsetting surfaces in the body.
                        The default value is PK_offset_method_accurate_c.
```

PK_FACE_identify_blends_o_t

```
struct PK_FACE_identify_blends_o_s
    {
    int                     o_t_version;      --- version number of options
                                              --- structure
    PK_double_constraint_t  limit_radii;      --- only return blends with radii
                                              --- that satisfy the constraint
                                              --- ( ( PK_comparison_always_c,
                                              ---     ( 0, 0 ) ) )
    PK_convexity_t          convexity;        --- only blends with the
                                              --- specified convexity will be
                                              --- returned.
                                              --- ( PK_convexity_unset_c )
    double                  tolerance;        --- tolerance (1.0e-6)
    PK_LOGICAL_t            want_radii;       --- whether to return the blends
                                              --- radii ( PK_LOGICAL_false )
    PK_LOGICAL_t            want_convexities; --- whether to return the blends
                                              --- convexities
                                              --- ( PK_LOGICAL_false )
    };
typedef struct PK_FACE_identify_blends_o_s PK_FACE_identify_blends_o_t;
```

The PK_FACE_identify_blends_o_t structure provides the following controls:

Description of fields:

```
'limit_radii'       The 'limit_radii' structure restricts which blend faces
                    are returned such that only blend faces with radii that
                    satisfy the limit(s) specified are returned.  The option
                    consists of an interval, and a comparison operation that
                    is used to compare the values with the interval.  The
                    default is to return all blends, regardless of their radii.

'convexity'         This option limits the blend faces returned such that only
                    blend faces of a certain convexity are returned.  The
                    default value is PK_convexity_unset_c.
                    'PK_convexity_arbitrary_c' will give the same behaviour as
                    'PK_convexity_unset_c', that is, blends will be
                    returned regardless of their convexity.

'want_radii'        If this flag is set to 'PK_LOGICAL_true' the radii of the
                    blend faces found will be returned.

'want_convexities'  If this flag is set to 'PK_LOGICAL_true' the convexities
                    of the blend faces found will be returned.
```

PK_FACE_identify_blends_r_t

```
struct PK_FACE_identify_blends_r_s
    {
    int                 n_blend_facesets;   --- number of blend facesets found
    PK_FACE_array_t    *blend_facesets;     --- faces classified as blends
```

```
    double             *blend_radii;        --- radii of blend facesets
    PK_convexity_t     *blend_convexities;  --- convexities of blend facesets
    };
typedef struct PK_FACE_identify_blends_r_s PK_FACE_identify_blends_r_t;
```

This structure contains the results of an identity blends operation. The PK memory associated with this structure may be freed by calling PK_FACE_identify_blends_r_f

Description of fields:

'n_blend_facesets' The number of blend facesets found

'blend_facesets' An array of blend facesets, each faceset contains an array of connected blend faces

'blend_radii' This field will be an array parallel to 'blend_facesets'. Each element of this array giving the radius of the blends in the corresponding element of 'blend_facesets'

'blend_convexities' This field will be an array parallel to 'blend_facesets'. Each element of this array gives the convexity of the blends in the corresponding element of 'blend_facesets'.

PK_FACE_imprint_cus_isoclin_o_t

```
struct PK_FACE_imprint_cus_isoclin_o_s
    {
    int            o_t_version;    --- version number of option structure
    PK_LOGICAL_t   old_iso;        --- whether former isocline curves are
                                   --- to be produced (PK_LOGICAL_false)
    PK_LOGICAL_t   all_points;     --- whether curves using all the control
                                   --- points are to be made (PK_LOGICAL_true)
    PK_LOGICAL_t   want_ret_faces; --- whether returned faces are required
                                   --- (PK_LOGICAL_true)
    PK_LOGICAL_t   want_is_steep;  --- whether steepness data are required
                                   --- (PK_LOGICAL_true)
    PK_LOGICAL_t   want_new_edges; --- whether new edges are required
                                   --- (PK_LOGICAL_true)
    };
typedef struct PK_FACE_imprint_cus_isoclin_o_s PK_FACE_imprint_cus_isoclin_o_t;
```

Contains optional controls on isocline curve generation and on returned arguments for function PK_FACE_imprint_cus_isoclin.

Description of fields:

'old_iso' If this flag is set to PK_LOGICAL_true, the former isocline curve behaviour is selected and the returned curves are identical to those produced using PK_FACE_imprint_curves_isocline. When this flag is set to PK_LOGICAL_false, the default value, the more accurate isocline curves are generated.
In the case of analytic isocline curves, this flag as no effect.

'all_points' In the general case, if this flag is set to PK_LOGICAL_true, the default value, all the control points which can be produced by the isocline calculation are used to generate the curves (NURBS vertices). Otherwise, if this flag is set to PK_LOGICAL_false, only half of them are used.
These latter curves are obviously less accurate

```
                        than those using all the control points, but
                        use less memory.
                        In the case of analytic isocline curves, this
                        flag is irrelevant.

'want_ret_faces'        If this flag is set to PK_LOGICAL_true, the
                        default value, the returned faces are required.

'want_is_steep'         If this flag is set to PK_LOGICAL_true, the
                        default value, the face steepness data will be
                        returned.

'want_edges'            If this flag is set to PK_LOGICAL_true, the
                        default value, the new edges will be returned.
```

PK_FACE_imprint_cus_normal_o_t

```
struct PK_FACE_imprint_cus_normal_o_s
    {
     int                o_t_version;  --- version number of option structure
    };
typedef struct PK_FACE_imprint_cus_normal_o_s PK_FACE_imprint_cus_normal_o_t;
```

The option structure defines options applicable to imprinting a set of curves on a set of faces.

PK_FACE_imprint_cus_vector_o_t

```
struct PK_FACE_imprint_cus_vector_o_s
    {
     int                o_t_version;  --- version number of option structure
    };
typedef struct PK_FACE_imprint_cus_vector_o_s PK_FACE_imprint_cus_vector_o_t;
```

The option structure defines options applicable to imprinting a set of curves on a set of faces.

PK_FACE_imprint_o_t

```
struct PK_FACE_imprint_o_s
    {
    int                 o_t_version;         --- version number
    PK_LOGICAL_t        imprint_tool;        --- (PK_LOGICAL_true)
    PK_LOGICAL_t        imprint_overlapping; --- (PK_LOGICAL_false)
    PK_LOGICAL_t        extend_face_list;    --- (PK_LOGICAL_false)
    const PK_boolean_match_o_t
                        *matched_region;     --- (NULL)
    };

typedef struct PK_FACE_imprint_o_s PK_FACE_imprint_o_t;
```

This option structure is supplied to the face imprinting operation.

Description of fields:

```
'imprint_tool'          Imprint edges and vertices on the tool faces.
                        If this flag is set to PK_LOGICAL_false,
                        edges and vertices are imprinted on the
                        target only. The arrays of edges and
                        vertices on the tool, returned in the results
                        structure, will be set to null.
```

'imprint_overlapping'	Imprint the boundaries of overlapping areas. This is only necessary when the target and tool are faces or sheet bodies. If a face from the target and a face from the tool are on the same surface, the boundaries of the overlapping area will be imprinted. If this flag is not set, faces on a common surface will not imprint on each other.
'extend_face_list'	Extend face list on the target. If the imprinting results in incomplete loops of imprinted edges, additional faces from the target will be intersected with the tool and imprinted in an attempt to form complete loops. No additional faces in tool will be used.

PK_FACE_inst_tools_r_t

```
struct PK_FACE_inst_tools_r_s
    {
    int                    n_instances;
    int                   *instance_indices;
    int                    n_tracked_faces;
    PK_FACE_tracking_r_t  *tracked_faces;
    };
typedef struct PK_FACE_inst_tools_r_s PK_FACE_inst_tools_r_t;
```

This data structure indicates which of the instanced tools appear in the resultant body.

Description of fields:

'n_instances'	The number of instanced tools which appear in the resultant body, this may be less than the number of supplied transforms, if for example one or more instanced tools miss the target.
'instance_indices'	An integer array of length 'n_instances' which indicates those instances which appear in the resultant body. The instance corresponding to the first transform in the given array has index value 0, the second has index value 1 and so on.

The following two fields are optionally returned.

'n_tracked_faces'	The number of faces in the tool body which are instanced at least once in the resultant body.
'tracked_faces'	An array of structures of type PK_FACE_tracking_r_t, one structure for each face in the tool body which is instanced at least once in the resultant body.

The function PK_FACE_inst_tools_r_f may be used to free associated memory.

PK_FACE_intersect_face_o_t

```
struct PK_FACE_intersect_face_o_s
    {
    int             o_t_version;    --- version number of option structure
    PK_LOGICAL_t    have_box;       --- whether box provided (PK_LOGICAL_false)
    PK_BOX_t        box;            --- box of interest
    PK_LOGICAL_t    have_uvbox_1;   --- whether box provided for face_1
                                    --- (PK_LOGICAL_false)
```

```
    PK_UVBOX_t      uvbox_1;        --- uvbox for face_1
    PK_LOGICAL_t    have_uvbox_2;   --- whether box provided for face_2
                                    --- (PK_LOGICAL_false)
    PK_UVBOX_t      uvbox_2;        --- uvbox for face_2
    PK_LOGICAL_t    have_vector;    --- whether seed vector provided
                                    --- (PK_LOGICAL_false)
    PK_VECTOR_t     vector;         --- only return branch containibg the seed
    };

typedef struct PK_FACE_intersect_face_o_s PK_FACE_intersect_face_o_t;
```

Holds optional controls on intersections between two faces.

The option structure defines options applicable to the intersection of two faces.

PK_FACE_intersect_surf_o_t

```
struct PK_FACE_intersect_surf_o_s
    {
    int             o_t_version;    --- version number of option structure
    PK_LOGICAL_t    have_box;       --- whether box provided (PK_LOGICAL_false)
    PK_BOX_t        box;            --- box of interest
    PK_LOGICAL_t    have_uvbox_1;   --- whether box provided for face
                                    --- (PK_LOGICAL_false)
    PK_UVBOX_t      uvbox_1;        --- uvbox for face
    PK_LOGICAL_t    have_uvbox_2;   --- whether box provided for surf
                                    --- (PK_LOGICAL_false)
    PK_UVBOX_t      uvbox_2;        --- uvbox for surf
    PK_LOGICAL_t    have_vector;    --- whether seed vector provided
                                    --- (PK_LOGICAL_false)
    PK_VECTOR_t     vector;         --- only return branch containibg the seed
    };

typedef struct PK_FACE_intersect_surf_o_s PK_FACE_intersect_surf_o_t;
```

Holds optional controls on intersections between a face and a surface.

The option structure defines options applicable to the intersection of a face and a surface.

PK_FACE_is_coincident_o_t

```
struct PK_FACE_is_coincident_o_s
    {
    int             o_t_version;
    };

typedef struct PK_FACE_is_coincident_o_s PK_FACE_is_coincident_o_t;
```

It currently has no application-settable fields.

PK_FACE_make_blend_o_t

```
struct PK_FACE_make_blend_o_s
    {
    int                     o_t_version;    --- version number of options
    PK_blend_trim_t         trim;           --- blend trimming option
```

```
                                          --- (PK_blend_trim_to_walls_c)
    PK_blend_walls_t        walls;        --- wall trimming option
                                          --- (PK_blend_walls_attach_c)
    PK_blend_shape_t        shape;        --- shape of blend
    PK_blend_constraint_t   constraints;  --- constraints on blend
    PK_blend_rib_control_t  rib_control;  --- blend section requests
    double                  tolerance;    --- tolerance (1.0e-05)
    PK_LOGICAL_t            have_help_point; --- (PK_LOGICAL_false)
    PK_VECTOR_t             help_point;   --- help point
    PK_LOGICAL_t            multiple;     --- create multiple blend sheets
                                          --- (PK_LOGICAL_false)
    PK_blend_propagate_t    propagate;    --- whether to propagate
                                          --- (PK_blend_propagate_no_c)
    PK_LOGICAL_t            notch;        --- enable blends to notch
                                          --- (PK_LOGICAL_false)
    PK_LOGICAL_t            local_check;  --- whether local checks required
                                          --- (PK_LOGICAL_false)
    PK_blend_local_check_t  checks;       --- which local checks to perform
    PK_blend_update_t       update;       --- disable changes for updates
                                          --- (PK_blend_update_1_c)
    PK_SURF_t               user_surface; --- user supplied blend surface
                                          --- (PK_ENTITY_null)
    };

typedef struct PK_FACE_make_blend_o_s PK_FACE_make_blend_o_t;
```

This structure contains options for face face blending.

The PK_FACE_make_blend_o_t contains the following options:

Trimming Options.

'trim' this option can have the values:

PK_blend_trim_no_c	Do not trim blend. The blend will not be trimmed to the wall extent. Its end boundaries will be constant parameter lines.
PK_blend_trim_short_c	Short trim to walls. The blend end boundaries will be the constant parameter lines determined by the wall boundaries such that the blend is as short as possible.
PK_blend_trim_long_c	Long trim to walls. The blend end boundaries will be the constant parameter lines determined by the wall boundaries such that the blend is as long as possible.
PK_blend_trim_to_walls_c	Trim blend to walls. The blend end boundaries will be determined by the wall boundaries. (default)

'walls' this option can have the values:

PK_blend_walls_trim_no_c	Do not trim walls. The walls will not be trimmed. The blend will not be attached to the walls but will be created as a separate sheet body.
PK_blend_walls_trim_both_c	Trim both walls. 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.

            PK_blend_walls_attach_c Trim walls and attach blend.
                                    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
                                    'left_sense' and 'right_sense' are
                                    not the same, the body containing the right
                                    wall will be negated in order to produce
                                    a consistent result.
                                    (default)

            PK_blend_walls_solid_c  Trim walls, attach blend and make solid
                                    (as above) and create solid body if possible
                                    If this option 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.

For further details on the use of these tokens please refer to the Parasolid
Functional Description Chapter "Face-face Blending".

                    Blend Shape Options.

 'xsection'   Cross section plane, may be one of:
              PK_blend_xs_rolling_ball_c (default) or
              PK_blend_xs_disc_c or
              PK_blend_xs_isoparameter_c

 'radius'     Radius of constant radius blend. (default 0.0)

 'parameter'  3D parameter curve

 'range1'     1D law function; ranges off left

 'range2'     1D law function; ranges off right

 'rho'        1D law function; rhos for general conics

 'depth'      2D law curve for curvature continuous blends

 'width'      Width of blend for defining constant width blends.

 'ratio'      Ratio of blend ranges for constant width blends.  The ratio is
              defined as "ranges off left" / "ranges off right".  The default
              is 1.0 i.e. equal ranges.

 'xs_shape'   Cross section shape, may be one of:
              PK_blend_xs_shape_unset_c  (default)
              PK_blend_xs_shape_conic_c
              PK_blend_xs_shape_g2_c
              PK_blend_xs_shape_chamfer_c

For further details on the use of these options please refer to the Parasolid
Functional Description Chapter "Face-face Blending".

                    Blend Constraint Options:

 'cliff_edges'    Blend is constrained by cliff edges.
                  An array of edge tags is given.
```

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

'tangent_edges' Blend is constrained by tangent hold lines.
An array of edge tags is given.
The blend will be constrained to be a tangent hold line blend in the region of the given edges.
For constant and variable radius rolling ball or disc blends the blend need not be constrained by these edges. If the blend is a variable radius rolling ball or disc blend, the left and right ranges of the law curve must be equal.

'conic_edges' Blend is constrained by conic hold lines.
An array of edge tags is given.
The blend will be constrained to be a conic hold line blend in the region of the given edges.
For constant and variable radius rolling ball blends blend need not be constrained by these edges. If the blend is a curvature continuous blend then 'cliff_edges' and 'tangent_edges' must not be set, and the number of edge tags in the array must be at least two.

'inv_tangent_edges'
Blend is constrained by inverted tangent hold lines.
An array of edge tags is given.
The blend will be constrained to be a tangent hold line blend in the region of the given edges.
For constant and variable radius rolling ball or disc blends the blend need not be constrained by these edges. If the blend is a variable radius rolling ball or disc blend, the left and right ranges of the law curve must be equal.

'inv_conic_edges'
Blend is constrained by inverted conic hold lines.
An array of edge tags is given.
The blend will be constrained to be a conic hold line blend in the region of the given edges.
For constant and variable radius rolling ball blends blend need not be constrained by these edges. If the blend is a curvature continuous blend then 'cliff_edges' and 'tangent_edges' must not be set, and the number of edge tags in the array must be at least two.

'limit_1'
'limit_2' Limit Planes.
One or two planes may be specified.
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.

For further details on the use of these options please refer to the Parasolid Functional Description Chapter "Face-face Blending".

Blend Rib Controls.

'output_rib' Control how ribs are produced.
This token controls under what circumstances face-face blending will produce ribs. The default is to not output ribs.

'group_rib' Control how to group the ribs produced.
Grouping controls how the limit on the number of ribs is applied. This token defines whether the ribs are grouped by blend face, or by the intervals along the 'parameter' curve defined by 'interval_parms', or not at all (default).

```
 'max_n_ribs'       Limit on number of ribs produced per group.
                    The number of ribs produced in each group will be no
                    greater than this number. The default value is zero,
                    for which no ribs will be produced.

 'n_interval_parms' Number of 'interval_parms' supplied.
                    Default is zero.

 'interval_parms'   Interval parameters.
                    These are used to divide the 'parameter' curve into
                    intervals for grouping ribs.  They should have values
                    in the parameter range of the curve.

For further details on the use of these options please refer to the Parasolid
Functional Description Chapter "Face-face Blending".

                         Control Options.

 'tolerance'        A tolerance is associated with this blend.

 'have_help_point' Whether a help point is provided

 'help_point'       Help point provided.
                    A vector, 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.

 'multiple'         Create multiple blends sheets.
                    PK_FACE_make_blend will only create one blend unless
                    this option is set to PK_LOGICAL_true, in which case
                    it will create all blends which lie in at least one of the
                    master faces.

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

 'notch'            End Overflow
                    If this options is set to PK_LOGICAL_true the blend will
                    be allowed to extend past notches, whether in the middle
                    or at the end of the blend.

 'local_check'      If this option is set to 'PK_LOGICAL_true', then
                    'checks' will be ignored, and the level of local checking
                    will be:

                         Surface self-intersection checks on all surfaces
                         except B-surfaces will be performed
                         Face checks will be performed
                         Face-face inconsistency checks will be perfomed.

                    If this option is set to 'PK_LOGICAL_false',
                    then the amount of local checking will
                    be determined by 'checks'.

 'checks'           Controls the amount of checking required. Ignored unless
                    'local_check' is set to 'PK_LOGICAL_false'.

 'update'           This option allows the disabling of some changes to the
                    blending algorithms for rebuild purposes. It should be set
                    to 'PK_blend_update_0_c' for updating v9.1 models.

 'user_surface'     This option allows users to users to supply a blend surface
                    to the face-face blending algorithm. The surface must be
                    within tolerance of the faces being blended. Parasolid
                    will attempt to use a given surface as a blend surface even
                    if this appears to be inconsistent with the 'left_sense' and
                    'right sense' arguments.
```

For further details on the use of these options please refer to the Parasolid Functional Description Chapter "Face-face Blending".

PK_FACE_neutral_causes_array_t

```
struct PK_FACE_neutral_causes_array_s
    {
    int          n_causes;    --- Number of face causes in array
                              --- (equal to number of faces in neutral sheet)
    PK_FACE_neutral_causes_t
                *face_causes; --- Set of face causes
    };
typedef struct PK_FACE_neutral_causes_array_s PK_FACE_neutral_causes_array_t;
```

This data structure contains an array of PK_FACE_neutral_causes_t structures.

Each element of the array contains the tracking information for one face of the coreesponding neutral sheet. The length of the array is 'n_causes' and this is the number of faces in the corresponding neutral sheet.

PK_FACE_neutral_causes_t

```
struct PK_FACE_neutral_causes_s
    {
    PK_FACE_t    neutral_face;   --- Face in neutral sheet.
    int          n_faces_back;   --- Number of faces compressed onto
                                 --- neutral_face from behind.
    PK_FACE_t   *faces_back;     --- Set of faces compressed onto
                                 --- neutral_face from behind.
    int          n_faces_front;  --- Number of faces compressed onto
                                 --- neutral_face from in front.
    PK_FACE_t   *faces_front;    --- Set of faces compressed onto
                                 --- neutral_face from in front.
    };
typedef struct PK_FACE_neutral_causes_s PK_FACE_neutral_causes_t;
```

This data structure contains the tracking information for a face in a neutral sheet.

A face on a neutral sheet is usually the result of intersecting the images of the projections of two faces from the corresponding face set pair, one from the first set and one from the the second set. There are cases, however, in which the number of faces which project to the neutral sheet is zero or two or more. The faces listed in 'faces_back' are the faces from the second set which project onto 'neutral_face' and those listed in 'faces_front' are the faces in the first set which project onto 'neutral_face'.

PK_FACE_offset_o_t

```
struct PK_FACE_offset_o_s
    {
    int                o_t_version;     --- version number of option structure
    PK_LOGICAL_t       allow_disjoint;  --- whether disjoint body can be
                                        --- produced
                                        --- (PK_LOGICAL_false)
    PK_check_fa_fa_t   check_fa_fa;     --- check for face-face inconsistencies
                                        --- (PK_check_fa_fa_yes_c)
    PK_offset_method_t offset_method;   --- offset method
```

```
                                        --- (PK_offset_method_accurate_c)
    PK_offset_step_t   offset_step;     --- whether to create a step offset
                                        --- (PK_offset_step_no_c)
    };
typedef struct PK_FACE_offset_o_s PK_FACE_offset_o_t;
```

This structure contains optional controls for PK_FACE_offset

Description of fields:

'allow_disjoint' — If this flag is set to PK_LOGICAL_true, and if the resulting body is split by the operation, all the disjoint components of that body will be preserved within a single body. If this flag is set to PK_LOGICAL_false, only one disjoint part of the body will be returned.
The default value is PK_LOGICAL_false.

'check_fa_fa' — Check for face-face inconsistencies. Permitted values are:
PK_check_fa_fa_yes_c
PK_check_fa_fa_no_c
The default value is PK_check_fa_fa_yes_c.

'offset_method' — The method used for offsetting surfaces in the body. The default value is PK_offset_method_accurate_c.

'offset_step' — A "boundary edge" is an edge between a face which is being offset and one which is not. If this option is set to PK_offset_step_yes_c then side faces will be created along any smooth boundary edges. Setting this option to PK_offset_step_yes_c has no effect on the result if there are no smooth boundary edges.
The default value is PK_offset_step_no_c.

PK_FACE_output_surf_trimmed_o_t

```
struct PK_FACE_output_surf_trimmed_o_s
    {
    int                     o_t_version;     --- option structure version
    --- surface options
    PK_FACE_trim_surf_t     trim_surf;       --- what surface to use
                                             --- (PK_FACE_trim_surf_own_c)
    PK_LOGICAL_t            cubic;           --- bsurf to be cubic
                                             --- (PK_LOGICAL_false)
    PK_LOGICAL_t            non_rational;    --- bsurf to be non-rational
                                             --- (PK_LOGICAL_false)
    double                  surf_tolerance;  --- tolerance for creating bsurf
    PK_LOGICAL_t            extend_surf;     --- is surface extension allowed
                                             --- (PK_LOGICAL_true)
    --- curve options
    PK_FACE_trim_confine_t  confine;         --- confinement
                                             --- (PK_FACE_trim_confine_closed_c)
    double                  curve_tolerance; --- tolerance for creating spcurves
                                             --- (0.0001)
    PK_FACE_trim_degen_t    degen;           --- in/exclude degeneracies
                                             --- (PK_FACE_trim_degen_no_c)
    --- output options
    PK_LOGICAL_t            want_geoms;      --- whether geometry required
```

```
                                                --- (PK_LOGICAL_false)
    PK_LOGICAL_t           want_topols;         --- whether topology required
                                                --- (PK_LOGICAL_false)
    };

typedef struct PK_FACE_output_surf_trimmed_o_s
               PK_FACE_output_surf_trimmed_o_t;
```

This option structure is used to select how a trimmed surface representation of a face is to be constructed.

The options are divided into Surface Options, SP-curve Options and Output Options.

Surface 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.

'trim_surf' : PK_FACE_trim_surf_own_c Use surface attached to face (default)
PK_FACE_trim_surf_bsurf_c Use a B-spline approximation

'extend_surf' : allow extension of the surface to fit SP-curves that stray outside.
(default = PK_LOGICAL_true)

When 'trim_surf' is set to PK_FACE_trim_surf_bsurf_c 3 other options may also be set:

'surf_tolerance': maximum allowed distance between the face's surface and the B-surface approximation.
(default = 0.00001)

'cubic' : Force output of B-spline of degree 3

'non_rational' : Force output of non-rational B-spline

No matter which type of surface is requested for output, PK_FACE_output_surf_trimmed will by default extend it to fit SP-curves approximating the face boundaries that stray outside of the natural boundaries of the surface. The field 'extend_surf' is provided to switch off this default behaviour. When 'extend_surf' is set to PK_LOGICAL_false, PK_FACE_output_surf_trimmed 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 negating 'extend_surf' may lead to output where SP-curves pass outside the natural boundaries of the surface on which they lie, any application setting this field to PK_LOGICAL_false must be able to deal with them.

SP-curve Options
-------- -------
There are 3 options available for the SP-curves which are to be constructed.

'curve_tolerance' : Allows specification of the tolerance which the SP-curve representations of the edges should satisfy.
(default = 0.0001)

'confine' : selects the trimmimg configuration.
See documentation for PK_FACE_trim_confine_t.
(default PK_FACE_trim_confine_closed_c)

'degen' : selects whether to represent surface degeneracies.
See documentation for PK_FACE_trim_degen_t.
(default PK_FACE_trim_degen_no_c)

As indicated above the SP-curve tolerance may be specified, this is a distance tolerance in model units and refers to the maximum allowable distance between an SP-curve and the edge curve that it represents. Although extensive tolerance 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 'surf_tolerance' should be no greater than half the 'curve_tolerance'.

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 PK_ERROR_tolerances_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 the function to be nothing more than a wire. In such cases, rather than output degenerate loops, the error PK_ERROR_trim_loop_degenerate will be returned. Setting a smaller tolerance may result in a successful attempt to produce trimmed surface output.

Output Options
------ -------

There are just two options related to the quantity of data returned. Each option is provided to indicate that the output arrays holding information associated with a trim loop is required.

'want_geoms' : Requests the return of the array of geometries ('geoms') and bounding intervals ('intervals') associated with the SP-curves in the PK_SURF_trim_data_t. (default = PK_LOGICAL_false)

'want_topols' : Requests the return of the list of model topologies ('topols') associated with the SP-curves in the PK_SURF_trim_data_t. (default = PK_LOGICAL_false)

PK_FACE_pattern_data_r_t

```
struct PK_FACE_pattern_data_r_s
    {
    PK_pattern_status_t        status;
    int                        n_pattern_faces;
    PK_FACE_t                 *pattern_faces;
    int                        n_error_topols;
    PK_TOPOL_t                *error_topols;
    };
typedef struct PK_FACE_pattern_data_r_s PK_FACE_pattern_data_r_t;
```

This data structure contains the status and the patterning information for a single instance of the pattern.

Description of fields:

'status' This status field indicates the state of this instance of the pattern. Permitted values are :

PK_pattern_status_ok_c
PK_pattern_status_fail_c

PK_pattern_status_outside_fa_c
PK_pattern_status_outside_reg_c
PK_pattern_status_tf_failed_c
PK_pattern_status_face_clash_c

'n_pattern_faces' The length of the 'pattern_faces' array.

'pattern_faces' The faces of this instance of the pattern which originated from the input pattern faces. There is a one to one relationship between this array and the input array of pattern faces.
If this instance of the pattern operation has not completely failed, the number of elements in this array will be the same as the number of input pattern faces. In the case of the PK_FACE_pattern function, the value of 'n_pattern_faces'.

'n_error_topols' The number of topological entities in the 'error_topols' array.

'error_topols' If the 'status' field of this return structure indicates a failure, this array may contain topological entities which are associated with the failure.

PK_FACE_pattern_o_t

```
struct PK_FACE_pattern_o_s
    {
    int                        o_t_version;   --- version number
    PK_pattern_check_loops_t   check_loops;   --- (PK_pattern_check_loops_no_c)
    PK_pattern_check_fa_fa_t   check_fa_fa;   --- (PK_pattern_check_fa_fa_no_c)
    };
typedef struct PK_FACE_pattern_o_s PK_FACE_pattern_o_t;
```

This option structure is supplied to the face patterning functions.

Description of fields:

'check_loops' Indicates whether loops checks are to be performed on the pattern instances, permitted values are :

PK_pattern_check_loops_yes_c
PK_pattern_check_loops_no_c

'check_fa_fa' Indicates whether face/face checks are to be performed on the resultant pattern instances, permitted values are :

PK_pattern_check_fa_fa_yes_c
PK_pattern_check_fa_fa_no_c

PK_FACE_pattern_r_t

```
struct PK_FACE_pattern_r_s
    {
    PK_pattern_result_t        result;
    int                        n_patterns;
    PK_FACE_pattern_data_r_t  *patterns;
```

```
    int                          n_error_topols;
    PK_TOPOL_t                  *error_topols;
    };
typedef struct PK_FACE_pattern_r_s PK_FACE_pattern_r_t;
```

This data structure contains operation status and the patterning results.

Description of fields:

'result' — This status field indicates the overall state of the patterning operations. Permitted values are :

- PK_pattern_result_ok_c
- PK_pattern_result_partial_c
- PK_pattern_result_fail_c
- PK_pattern_result_mixed_faces_c
- PK_pattern_result_instanced_c
- PK_pattern_result_invalid_set_c
- PK_pattern_result_miss_geom_c

'n_patterns' — The length of the 'patterns' array.

'patterns' — The array of pattern instance results. If the 'result' field indicates an error or a partial error, the individual pattern result records (if returned) must be inspected. If the pattern operation has not completely failed, the number of elements in this array will be the same as the number of pattern instances requested. In the case of the PK_FACE_pattern function, the value of 'n_transforms'.

'n_error_topols' — The number of topological entities in the 'error_topols' array.

'error_topols' — If the 'result' field of this return structure indicates a failure, this array may contain topological entities which are associated with the failure.

The function PK_FACE_pattern_r_f may be used to free associated memory.

PK_FACE_repair_o_t

```
struct PK_FACE_repair_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        split_g1;       --- split at G1 discontinuities
                                        ---     (default PK_LOGICAL_true)
    PK_LOGICAL_t        split_self_int; --- split to avoid self-intersections
                                        ---     (default PK_LOGICAL_true)
    };

typedef struct PK_FACE_repair_o_s PK_FACE_repair_o_t;
```

This options structure holds optional controls for PK_FACE_repair.

Description of fields:

'split_g1' — Split the face along lines of G1 discontinuity.

'split_self_int' — Split the face so that the parametric ranges of the subdivided faces avoid any self-intersecting areas in the original surface.

PK_FACE_replace_surfs_o_t

```
struct PK_FACE_replace_surfs_o_s
    {
    int                         o_t_version; --- version number
    PK_check_fa_fa_t            check_fa_fa; --- check for face-face
                                             --- inconsistencies
                                             --- (PK_check_fa_fa_yes_c)
    PK_replace_edge_data_t      edge_data;   --- replace data for edges
    PK_replace_vertex_data_t    vertex_data; --- replace data for vertices
    PK_replace_help_points_t    edge_help;   --- edge help points
    PK_replace_help_points_t    vertex_help; --- vertex help points
    PK_replace_merge_t          merge;       --- whether changed faces are to
                                             --- be merged with their neighbours
                                             --- if feasible
                                             --- (PK_replace_merge_no_c)
    };

typedef struct PK_FACE_replace_surfs_o_s PK_FACE_replace_surfs_o_t;
```

This option structure is supplied to the function to replace the geometry of faces.

Description of fields:

'check_fa_fa' — Check for face-face inconsistencies. Permitted values are :

- PK_check_fa_fa_no_c
- PK_check_fa_fa_yes_c

The default value is PK_check_fa_fa_yes_c

'edge_data' — Structure containing the data relative to the edge curve replacements. Refer to its documentation for details.

'vertex_data' — Structure containing the data relative to the vertex point replacements. Refer to its documentation for details.

'edge_help' — Structure containing the data relative to the edge help points. Help points are used to pick new edge curves. In case of multiple solutions, the closest curve to the related help point is chosen.

'vertex_help' — Structure containing the data relative to the vertex help points. Help points are used to pick new vertex positions. In case of multiple solutions, the closest point to the related help point is chosen.

'merge' — Adjacent faces may become mergeable when the surface of one or both faces is replaced. Permitted values are :

- PK_replace_merge_no_c
- PK_replace_merge_in_c
- PK_replace_merge_out_c

The default value is PK_replace_merge_no_c

PK_FACE_replace_surfs_r_t

```
struct PK_FACE_replace_surfs_r_s
    {
    PK_local_status_t          status;
    int                        n_error_entities;
    PK_ENTITY_t                *error_entities;
    };
typedef struct PK_FACE_replace_surfs_r_s PK_FACE_replace_surfs_r_t;
```

This data structure contains the status of the operation and any error information if a failure is indicated.

Description of fields:

'status' This status field indicates the overall state of the local operation. Permitted values are :

- PK_local_status_ok_c
- PK_local_status_nocheck_c
- PK_local_status_fail_c
- PK_local_status_cant_get_pt_c
- PK_local_status_cant_get_cu_c
- PK_local_status_cant_get_su_c
- PK_local_status_cant_offset_c
- PK_local_status_fa_fail_c
- PK_local_status_fa_fa_fail_c
- PK_local_status_ed_remains_c

'n_error_entities' The number of topological or geometric entities in the 'error_entities' array.

'error_entities' If the 'status' field of this return structure indicates a failure, this array may contain topological or geometric entities which are associated with the failure.

Error entities returned

The 'error_entities' array will contain the following entities according to the value of the 'status' field :

PK_local_status_ok_c	: No entities		
PK_local_status_nocheck_c	: No entities		
PK_local_status_cant_get_pt_c	: 1 VERTEX		Failed to find new geometry for existing vertex.
	n FACEs		Failed to find geometry for new vertex.
PK_local_status_cant_get_cu_c	: 1 EDGE		Failed to find new geometry for existing edge.
	n FACEs		Failed to find geometry for new edge.
PK_local_status_cant_get_su_c	: n FACEs		Unable to produce offset surfaces for these faces. No surface replacement is performed.
PK_local_status_cant_offset_c	: 1 FACE		Failed to offset or face could not be deleted.
PK_local_status_fa_fail_c	: 1 FACE		Face which failed check.
PK_local_status_fa_fa_fail_c	: 2 FACEs		Pair of faces where inconsistency found.

```
  PK_local_status_ed_remains_c  : 1 EDGE     An existing edge should have
                                             disappeared
                                  n FACEs    A transient edge should have
                                             disappeared

  PK_local_status_fail_c        : No entities
```

PK_FACE_section_o_t

```
struct PK_FACE_section_o_s
    {
    int                         o_t_version;      --- version number
    PK_section_fence_t          fence;            --- (PK_section_fence_both_c)
    const PK_boolean_config_o_t
                               *configuration;    --- (NULL)
    const PK_boolean_region_select_o_t
                               *select_region;    --- (NULL)
    const PK_boolean_match_o_t
                               *matched_region;   --- (NULL)
    PK_LOGICAL_t                extend_face_list; --- (PK_LOGICAL_false)
    PK_LOGICAL_t                merge_imprinted;  --- (PK_LOGICAL_false)
    PK_LOGICAL_t                merge_new_faces;  --- (PK_LOGICAL_false)
    PK_LOGICAL_t                selective_merge;  --- (PK_LOGICAL_false)
    PK_section_check_fa_t       check_fa;         --- (PK_section_check_fa_yes_c)
    double                      default_tol;      --- (1.0E-5)
    double                      max_tol;          --- (0.0)
    PK_LOGICAL_t                tracking;         --- (PK_LOGICAL_false)
    PK_LOGICAL_t                keep_target_edges;  --- (PK_LOGICAL_false)
    };
typedef struct PK_FACE_section_o_s PK_FACE_section_o_t;
```

This option structure is supplied to a section operation for bodies.

Description of fields:

'fence' — To determine which resultant bodies or faces are returned when performing a sectioning operation. Permitted values are :
- PK_section_fence_front_c
- PK_section_fence_back_c
- PK_section_fence_both_c

'configuration' — Describes the configuration of target and tool bodies.

'select_region' — Used to identify regions of the tool to include or exclude in the result.

'matched_region' — Describes matched regions between the target face and the tool sheet.

'extend_face_list' — A face section receives a subset of the faces of a target body. If the imprinting phase of the section results in incomplete loops of imprinted edges, then additional target faces will be used.

'merge_imprinted' — Merge all mergeable imprinted edges created and/or located by the section operation.
Note the following :
- o an existing edge can be considered an imprinted edge if it forms part of the intersection between faces from the target and tool.
- o mergeable edges which are adjacent to new section faces are not merged (this behaviour is particular to sectioning).

'merge_new_faces'	Merge all mergeable imprinted edges created by the section operation including edges which are adjacent to new section faces. This flag has no effect if the 'merge_imprinted' field is not set to PK_LOGICAL_true.
'selective_merge'	If this is set to PK_LOGICAL_true, the merge phase will avoid mergeing imprinted edges that existed and were mergeable before the start of the section. This flag has no effect if 'merge_imprinted' is not set to PK_LOGICAL_true.
'check_fa'	The value of this determines if face checking is performed during the operation, permitted values are : PK_section_check_fa_no_c PK_section_check_fa_yes_c The default is set to 'PK_section_check_fa_yes_c' and it is strongly advised not to change this value. If the face checking is switched off, the section operation may produce an invalid body without raising an error.
'default_tol'	The default tolerance to which the sectioning operation, if it has to, may approximate entities or assume that entities are coincident. This value should be set to a value that is related to the tolerance to which the application is creating approximations. If this field is set to zero (0.0), it is assumed that the decision to set tolerances on entities is being left to the modeller.
'max_tol'	This value specifies the maximum tolerance which may be applied to any entity involved in the sectioning operation. If this field is set to zero (0.0), the default, then it assumed that no upper limit has been set on the tolerance of any entity. If the field is not set to zero (0.0), then it must be set to a value that is greater than the value of the 'default_tol' field.
'tracking'	Not currently in use.
'keep_target_edges'	This option controls which edge survives when a target edge and a tool edge coincide in a boolean. If both faces adjacent to the tool edge survive, then the tool edge will also survive. Otherwise, the result depends on the setting of the option: PK_LOGICAL_true: The target edge survives, unless the target edge is a new edge imprinted during the boolean, and the tool edge is an original edge - in this case, the tool edge survives. PK_LOGICAL_false (the default): The older edge (i.e. the one with the smaller tag) survives. This option only has an effect if general topology is disabled. If it is enabled, then the target edge will always survive, even if it is a newly imprinted edge.

PK_FACE_set_pair_t

```
struct PK_FACE_set_pair_s
    {
    int          n_faces1;   --- Number of faces in first set
    PK_FACE_t   *faces1;     --- First set of faces
    int          n_faces2;   --- Number of faces in second set
    PK_FACE_t   *faces2;     --- Second set of faces
    PK_BODY_t    sheet;      --- Sheet to be trimmed
    PK_LOGICAL_t sheet_pretrimmed;
                             --- Whether 'sheet' is already trimmed
    };
typedef struct PK_FACE_set_pair_s PK_FACE_set_pair_t;
```

This data structure contains a pair of sets of faces and its corresponding neutral sheet.

The faces in 'faces1' and 'faces2' will specify the way the sheet body given in the field 'sheet' should be trimmed. Each face set must contain at least one face and all the faces must belong to the same solid body. 'sheet' may contain any number of faces but all the faces must have the same surface. 'sheet' may not be PK_ENTITY_null.

The value of the logical field 'sheet_pretrimmed' indicates whether the neutral sheet is to be trimmed. If it is PK_LOGICAL_true, then trimming of the neutral sheet is not required but the definition of the face set pair is required to specify how adjacent neutral sheets are to be trimmed. If the value of 'sheet_pretrimmed' is PK_LOGICAL_false, then the neutral sheet is required to be trimmed.

PK_FACE_simplify_geom_o_t

```
struct PK_FACE_simplify_geom_o_s
    {
    int          o_t_version;  --- version number of option structure
    PK_LOGICAL_t want_geoms;   --- user requires new geoms to be returned
                               --- (Default is PK_LOGICAL_false)
    };

typedef struct PK_FACE_simplify_geom_o_s PK_FACE_simplify_geom_o_t;
```

This options structure will hold optional controls for PK_FACE_simplify_geom.

Description of fields:

'o_t_version'	version number of option structure.
'want_geoms'	user requires new geoms to be returned (Default is PK_LOGICAL_false).

PK_FACE_split_at_param_o_t

```
struct PK_FACE_split_at_param_o_s
    {
    int                o_t_version;    --- version number of option structure
    };

typedef struct PK_FACE_split_at_param_o_s PK_FACE_split_at_param_o_t;
```

This options structure will hold optional controls for PK_FACE_split_at_param.

Description of fields:

'o_t_version' version number of option structure.

PK_FACE_t

This type represents a FACE. A face is a bounded connected subset of a surface.

A face is a type of topology (PK_CLASS_face is a subclass of PK_CLASS_topol).

PK_FACE_taper_o_t

```
struct PK_FACE_taper_o_s
    {
    int               o_t_version;   --- version number of option structure
    PK_LOGICAL_t      merge_face;    --- whether tapered faces are to be merged
                                     --- with their neighbours
                                     --- (PK_LOGICAL_true)
    PK_check_fa_fa_t  check_fa_fa;   --- check for face-face inconsistencies
                                     --- (PK_check_fa_fa_yes_c)
    PK_EDGE_array_t   tapered_edges; --- tapered step edges
    PK_EDGE_array_t   normal_edges;  --- normal step edges
    PK_taper_method_t method;        --- main face taper method
                                     --- (PK_taper_method_isocline_c)
    double            offset;        --- offset value (0.0)
    PK_SURF_t         top_surface;   --- top surface (PK_ENTITY_null)
    };
typedef struct PK_FACE_taper_o_s PK_FACE_taper_o_t;
```

This structure contains optional controls for PK_FACE_taper

Description of fields:

'merge_face' If this flag is set to PK_LOGICAL_true, the new faces produced by the operation will be merged with their neighbouring faces, if it is feasible. The default value is PK_LOGICAL_true.

'check_fa_fa' Check for face-face inconsistencies.
Values can be :
    PK_check_fa_fa_no_c
    PK_check_fa_fa_yes_c
The default value is PK_check_fa_fa_yes_c.

'tapered_edges' Edges which will give rise to tapered step faces. These faces' surfaces will satisfy the taper condition. Edges in this array should NOT also appear in the 'normal_edges' array.

'normal_edges' Edges which will give rise to normal step faces. These faces' surfaces are ruled surfaces normal to the relevant original taper face.
These surfaces do NOT necessarily satisfy the taper condition. Edges in this array should NOT also appear in the 'tapered_edges' array.

'method' Taper method to construct main taper faces' surfaces.
Values can be :
    PK_taper_method_isocline_c : isocline method
    PK_taper_method_curve_c   : curve method
    PK_taper_method_surface_c : surface method
    PK_taper_method_offset_c   : offset method

	o The isocline (or silhouette) method produces ruled isocline surfaces satisfying the taper condition. It will fail if the angle between the reference curve tangent and the taper direction is smaller than the taper angle. The taper surfaces were previouly produced using this method. o The curve method produces ruled surfaces whose ruling direction is in the plane containing the taper direction and the direction perpendicular to the taper direction and the reference curve tangent. The reference curve is either the reference edge's curve or the reference surface and taper face intersection curve. o The surface method produces ruled surfaces whose ruling direction is in the plane containing the taper direction and the tapered face's surface normal. o The offset method produces ruled surfaces from reference edge's curves to their "offset curves". An "offset curve" is a curve on the given top surface and that appears offset the given distance from the relevant reference edge's curve when viewed from the taper direction. The default value is PK_taper_method_isocline_c.
'offset'	The offset distance for the offset taper method. Positive/negative offset distance makes the curve to offset towards the outside/inside of the body.
'top_surface'	The surface on which the reference edge's curve is projected to create the relevant offset curve.

PK_FACE_tracking_r_t

```
struct PK_FACE_tracking_r_s
    {
    PK_FACE_t   original_face;
    int         n_times_instanced;
    PK_FACE_t  *instanced_faces;
    int        *instance_index;
    };
typedef struct PK_FACE_tracking_r_s PK_FACE_tracking_r_t;
```

For a tool face which is instanced at least once in the resultant body, this structure gives those faces which are instanced copies of that face and which instance they belong to.

Description of fields:

'original_face'	The original face in the tool body (unmodified).
'n_times_instanced'	Total number of times instanced copies of this face appear in the resultant body, if a face fragments then each fragment has a separate entry.
'instanced_faces'	The instanced faces of 'original_face'.
'instance_index'	Indices of 'instanced_faces', faces appearing in the instanced copy corresponding to the first transform in the given array will have an index value of 0, those corresponding to the second transform will have an index value 1 and so on.

PK_FACE_transform_o_t

```
struct PK_FACE_transform_o_s
    {
    int              o_t_version;  --- version number of option structure
    PK_LOGICAL_t     merge_face;   --- whether transformed faces are to be
                                   --- merged with their neighbours,
                                   --- if feasible (PK_LOGICAL_true)
    PK_check_fa_fa_t check_fa_fa;  --- check for face-face inconsistencies
                                   --- (PK_check_fa_fa_yes_c)
    };
typedef struct PK_FACE_transform_o_s PK_FACE_transform_o_t;
```

This structure contains optional controls for PK_FACE_transform.

Description of fields:

'merge_face' — If this flag is set to PK_LOGICAL_true, the new faces produced by the operation will be merged with their neighbouring faces, if it is feasible. The default value is PK_LOGICAL_true.

'check_fa_fa' — Check for face-face inconsistencies.
Values can be :
    PK_check_fa_fa_no_c
    PK_check_fa_fa_yes_c
The default value is PK_check_fa_fa_yes_c.

PK_FACE_trim_confine_t

```
typedef int PK_FACE_trim_confine_t;
```

This data type is used to select how a trimmed surface is to be confined. It has four values:

PK_FACE_trim_confine_no_c	Trim 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. the ends of a cylinder). The SP-curves in a loop may have gaps between them in parameter space where there are degeneracies.
PK_FACE_trim_confine_yes_c	Trim 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 SP-curves in a loop may have gaps between them in parameter space where there are degeneracies.
PK_FACE_trim_confine_closed_c	Trim loops will be confined to a single period on periodic faces. All the trim 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.
PK_FACE_trim_confine_periph_c	Trim loops will be confined to a single period on periodic faces. All the trim loops will be closed and without gaps in parameter space. Each trimmed surface will have exactly one outer peripheral loop.

PK_FACE_trim_degen_t

typedef int PK_FACE_trim_degen_t;

This data type is used to select whether surface degeneracies will be included in the output of PK_FACE_output_surf_trimmed. It has two values:

PK_FACE_trim_degen_no_c	Do not include degeneracies except those implied by the PK_FACE_trim_confine_t setting.
PK_FACE_trim_degen_yes_c	All parametric degeneracies occurring on the face will be represented (irrespective of whether they are associated with topology).

PK_FACE_trim_surf_t

typedef int PK_FACE_trim_surf_t;

This data type is used to select what surface is to be used by PK_FACE_output_surf_trimmed. It has two values:

PK_FACE_trim_surf_own_c	use copy of face's own surface
PK_FACE_trim_surf_bsurf_c	use B-spline approximation of face's surf

PK_facet_cull_t

typedef int PK_facet_cull_t;

This field can take any one of the following values.

'PK_facet_cull_none_c'	: do not cull any facets; output them all
'PK_facet_cull_back_c'	: cull back facing facets (requires view)

PK_facet_degen_t

typedef int PK_facet_degen_t;

This field can take any one of the following values.

'PK_facet_degen_multiple_vxs_c'	: output multiple vertices at degeneracies
'PK_facet_degen_single_vx_c'	: output single vertex at degeneracies
'PK_facet_degen_average_parms_c'	: output average parameters at degeneracies

PK_facet_density_t

typedef int PK_facet_density_t;

This field can take any one of the following values.

'PK_facet_density_no_view_c'	: facet density is independent of view
'PK_facet_density_use_view_c'	: increase density at silh. (requires view)

PK_facet_fault_t

typedef int PK_facet_fault_t;

This typedef can take any one of the following values.

```
    'PK_facet_fault_face_c'          : face could not be facetted
    'PK_facet_fault_facet_c'         : facet disconnected from neighbours
    'PK_facet_fault_vertex_point_c'  : point data for vertex is not valid
    'PK_facet_fault_vertex_normal_c' : normal data for vertex is not valid
    'PK_facet_fault_vertex_param_c'  : param data for vertex is not valid
    'PK_facet_fault_thin_face_c'     : no facets output for thin face
    'PK_facet_fault_body_c'          : body could not be facetted
```

PK_facet_go_edges_t

typedef int PK_facet_go_edges_t;

This field can take any one of the following values.

```
    'PK_facet_go_edges_no_c'       : do not output edge data
    'PK_facet_go_edges_yes_c'      : output edge entities at facet edges
```

PK_facet_go_interleaved_t

typedef int PK_facet_go_interleaved_t;

The 'go_interleaved' option allows facets from any one of the bodies for which a body segment is open to be output. Facets are output on a face by face basis rather than a body by body basis, in cases where there are multiple instances selecting this option should reduce the peak memory of facetting.

Interleaved facetting has the additional advantage of being multi-threaded.

This field can take any one of the following values :

```
PK_facet_go_interleaved_no_c  : output all the facets for one body before
                                beginning to output facets from another body
                                (default).
PK_facet_go_interleaved_yes_c : facets are output on a face by face basis. In
                                cases where there are multiple instances or
                                multi-threading is turned on facets for a
                                particular face are output as soon as
                                possible so facets may be output from
                                different bodies consecutively.
```

PK_facet_go_normals_t

typedef int PK_facet_go_normals_t;

This field can take any one of the following values.

```
    'PK_facet_go_normals_no_c'     : do not output surface normals
    'PK_facet_go_normals_yes_c'    : output surface normals at facet vertices
```

PK_facet_go_parameters_t

typedef int PK_facet_go_parameters_t;

This field can take any one of the following values.

```
    'PK_facet_go_parameters_no_c'  : do not output surface parameter data
    'PK_facet_go_parameters_d0_c'  : output surface params at vertices
    'PK_facet_go_parameters_d1_c'  : output surface params + first derivatives
    'PK_facet_go_parameters_d2_c'  : output surface params + all second derivs
```

PK_facet_go_strips_t

```
typedef int PK_facet_go_strips_t;
```

This field can take any one of the following values.

```
    'PK_facet_go_strips_no_c'      : output individual facets through GO
    'PK_facet_go_strips_yes_c'     : output facets through GO in strips
```

PK_facet_ignore_scope_t

```
typedef int PK_facet_ignore_scope_t;
```

This field can take any one of the following values.

```
    'PK_facet_ignore_scope_local_c' : treat face as individual entity
    'PK_facet_ignore_scope_global_c': treat face as part of face's body
```

PK_facet_ignore_t

```
typedef int PK_facet_ignore_t;
```

This field can take any one of the following values.

```
    'PK_facet_ignore_no_c'        : no features ignored
    'PK_facet_ignore_absolute_c'  : input an absolute value
    'PK_facet_ignore_ratio_c'     : input the ratio of feature to model box
    'PK_facet_ignore_body_ratio_c': input the ratio of feature to body box
```

PK_facet_local_tolerances_t

```
struct PK_facet_local_tolerances_s
    {
    double             curve_chord_tol;
    double             curve_chord_max;
    double             curve_chord_ang;
    double             surface_plane_tol;
    double             surface_plane_ang;
    };
typedef struct PK_facet_local_tolerances_s PK_facet_local_tolerances_t;
```

Local facet tolerances

The PK_facet_local_tolerances_t structure provides the following controls:

CURVE TOLERANCES

The following arguments control the curve tolerance values, measuring how closely a curved edge entity is approximated by a number of straight line chords.

Three curve tolerance controls are provided:

curve_chord_tol : the maximum chordal distance between a facet edge and its original edge entity

curve_chord_max : the maximum length of any facet edge which represents part of an original edge entity

curve_chord_ang : the maximum angle which is permitted between a facet chord and its original edge entity

SURFACE TOLERANCES

The following arguments control surface tolerance values, measuring

how closely any curved surface attached to a face is approximated by a facet mesh.

Two surface tolerance controls are provided :

surface_plane_tol : the maximum distance between the mid-plane of a facet and its original face entity

surface_plane_ang : the maximum angle (in radians) which is permitted between the surface normals at any two positions on the surface which lie within the facet boundary (this is usually met but it is not guaranteed).

To set one or more of these constraints, the value is set to a positive value. If the value of a constraint is set to 0.0 the local tolerance of this constraint will not override the tolerance of the corresponding constraint previously supplied in the PK_TOPOL_facet_mesh_o_t. If a particular constraint is set to 0.0 and the value of the corresponding constraint in the PK_TOPOL_facet_mesh_o_t is not supplied the default for this constraint will be used.

PK_facet_match_t

typedef int PK_facet_match_t;

This field can take any one of the following values.

```
    'PK_facet_match_geom_c'        : clip facet boundaries to a common edge
    'PK_facet_match_topol_c'       : match facet vertices across a common edge
    'PK_facet_match_trimmed_c'     : clip facet boundaries to model edge curve
```

PK_facet_shape_t

typedef int PK_facet_shape_t;

```
    'PK_facet_shape_any_c'         : allow creation of holed or concave facets
    'PK_facet_shape_cut_c'         : cut facets with holes into concave facets
    'PK_facet_shape_convex_c'      : divide concave facets into convex facets
```

PK_facet_wire_edges_t

typedef int PK_facet_wire_edges_t;

This field can take any one of the following values.

```
    'PK_facet_wire_edges_no_c' : facet over wire edges in faces
    'PK_facet_wire_edges_yes_c': do not facet over wire edges in faces
```

PK_FCURVE_array_t

This structure contains an array of entities of type PK_FCURVE_t

```
typedef struct PK_FCURVE_array_s
    {
    PK_FCURVE_t    *array;   --- array of Foreign CURVEs
    int             length;  --- length of array
    }
    PK_FCURVE_array_t;
```

It has the fields:

'array': array of entities of type PK_FCURVE_t

'length': the number of entities in 'array'

PK_FCURVE_sf_t

```
struct PK_FCURVE_sf_s
    {
    char       *key;       --- curve key
    int         n_ints;    --- number of integer values (>=0)
    int        *ints;      --- integer values
    int         n_doubles; --- number of double values  (>=0)
    double     *doubles;   --- double values
    size_t      space;     --- data space required      (>=0)
    PK_TRANSF_t transf;    --- transformation (may be PK_ENTITY_null)
    };
typedef struct PK_FCURVE_sf_s PK_FCURVE_sf_t;
```

This structure is the standard form of a foreign curve.

Specific Errors

PK_ERROR_FG_eval_not_found	curve evaluator not found
PK_ERROR_FG_data_alloc_error	curve evaluator data allocation fault
PK_ERROR_FG_doubles	'doubles' data error
PK_ERROR_FG_ints	'ints' data error
PK_ERROR_FG_data_not_found	curve evaluator data not found
PK_ERROR_FG_evaluator_error	curve evaluator failure
PK_ERROR_FG_modelling_error	cannot model with this curve

A separate Foreign Geometry User Guide is available.

Sufficient data space is always allocated to store one double. The 'space' field indicates data space in addition to this.

PK_FCURVE_t

This type represents a Foreign CURVE; that is, one implemented in an externally supplied parametric evaluator.

A foreign curve is a type of curve (PK_CLASS_fcurve is a subclass of PK_CLASS_curve).

The fcurve is defined by data contained in a PK_FCURVE_sf_t

PK_FFCLOS_f_t

```
typedef void (*PK_FFCLOS_f_t)(const int *guise,
                              const int *strid, const int *action,
                                    int *ifail);
```

Close file

Receives:

```
  int                    *guise          --- class of file:
                                         ---  FFCROL,FFCSNP,FFCJNL,
                                         ---  FFCXMT,FFCXMO,FFCSCH,
```

```
                                          ---  FFCLNC,FFCDBG
  int                     *strid          --- frustrum strid
  int                     *action         --- type of close:
                                          ---  FFNORM,FFABOR

Returns:
  int                     *ifail          --- error code: FR_no_errors
                                          ---             FR_close_fail

Description:
  This function closes a frustrum file which has been opened with FFOPRD,
  FFOPWR or FFOPRB. In the first two cases, if the file is newly created,
  the calling routine can determine whether to retain the file (FFNORM)
  or to delete it (FFABOR). In the latter case, the rollback file is
  always deleted.
```

PK_FFOPRB_f_t

```
typedef void (*PK_FFOPRB_f_t)(const int *guise,
                              const int *minsiz, const int *maxsiz,
                                    int *actsiz, int *strid, int *ifail);

  Open rollback file. This function is documented for legacy
  purposes only, and should not be implemented for new Parasolid applications.

Receives:
  int                     *guise          --- class of file: FFCROL (rollback)
  int                     *minsiz         --- minimum size of file (bytes)
  int                     *maxsiz         --- maximum size of file (bytes)

Returns:
  int                     *actsiz         --- actual size of file (bytes)
  int                     *strid          --- frustrum strid
  int                     *ifail          --- error code: FR_no_errors
                                          ---             FR_open_fail
                                          ---             FR_disc_full

Description:
  This function opens a new binary rollback file, returning a frustrum stream
  identifier or 'strid' for use by FFREAD, FFWRIT, FFTELL, FFSEEK and FFCLOS.
  The file is allocated a length of between 'minsiz' and 'maxsiz' bytes, the
  actual size being returned in 'actsiz'. The actual size is normally
  'maxsiz' bytes, but may be reduced, for example, to give a whole number of
  disc blocks or if insufficient disc space is available.
```

PK_FFOPRD_f_t

```
typedef void (*PK_FFOPRD_f_t)(const int *guise,
                              const int *format, const char name[],
                              const int *namlen, const int *skiphd,
                                    int *strid, int *ifail);

  Open all guises of file (except rollback) for reading

Receives:
  int                     *guise          --- class of file:
                                          ---  FFCSNP,FFCJNL,FFCXMT
                                          ---  FFCXMO,FFCSCH,FFCLNC
                                          ---  snapshot,journal,C-transmit,
                                          ---  Fortran-transmit,schema,licence
  int                     *format         --- format code:
                                          ---  FFBNRY,FFTEXT (binary or text)
  char                     name[]         --- key which identifies file
                                          ---  (terminator not required)
```

```
  int                     *namlen         --- length of name
  int                     *skiphd         --- action required on opening file
                                          ---  FFSKHD,FFLVHD
                                          ---  skip header (the usual case) or
                                          ---  leave header (used by Frustrum
                                          ---                acceptance tests)

Returns:
  int                     *strid          --- id for stream on which
                                          --- file is open
  int                     *ifail          --- error code: FR_no_errors
                                          ---             FR_bad_name
                                          ---             FR_not_found
                                          ---             FR_bad_header
                                          ---             FR_open_fail

Description:
  This function opens all guises of existing files (except rollback) for
  reading, i.e. schema, C and Fortran transmit, journal, licence and snapshot
  files.

  If the 'skiphd' flag is set to FFSKHD, the header data is skipped when
  the file is opened. This is the mode which is used by Parasolid.

  If the 'skiphd' flag is set to FFLVHD, the header data is not skipped
  when the file is opened; the preamble, the parts data and the trailer
  are read by subsequent calls to FFREAD. This mode is only used by the
  TESTFR to validate what has been written by a particular implementation.

  The routine returns a frustrum stream identifier or 'strid'.
  This is used in subsequent calls to FFREAD and FFCLOS.
```

PK_FFOPWR_f_t

```
typedef void (*PK_FFOPWR_f_t)(const int *guise,    const int *format,
                              const char name[],   const int *namlen,
                              const char pr2hdr[], const int *pr2len,
                                    int *strid, int *ifail);

  Open all guises of file (except rollback) for writing

Receives:
  int                     *guise          --- class of file:
                                          ---  FFCSNP,FFCJNL,FFCXMT,
                                          ---  FFCSCH,FFCLNC,FFCDBG
                                          ---  snapshot,journal,C-transmit,
                                          ---  schema,licence,debug report
  int                     *format         --- format code:
                                          ---  FFBNRY,FFTEXT,FFXML
                                          ---  (binary,text,XML)
  char                     name[]         --- key which identifies file
                                          ---  (terminator not required)
  int                     *namlen         --- length of name
  char                     pr2hdr[]       --- part 2 header data from
                                          --- Parasolid
                                          ---  (terminator not required)
  int                     *pr2len         --- length of part 2 header data

Returns:
  int                     *strid          --- id for stream on which
                                          --- file is open
  int                     *ifail          --- error code: FR_no_errors
                                          ---             FR_bad_name
                                          ---             FR_already_exists
                                          ---             FR_open_fail
```

```
                                          ---            FR_write_fail
                                          ---            FR_disc_full

Description:
  This function opens all guises of new files (except rollback) for
  writing, i.e. schema, C transmit, journal snapshot, licence files and
  debug report files.

  The 'pr2hdr' string contains a sequence of keyword names and values in the
  form KEYWORD1=value1;KEYWORD2=value2 etc. These describe the implementation
  of Parasolid, for storage in the file header. The characters in the keyword
  values do not need to be escaped when they are written to file.

  The routine returns a frustrum stream identifier or 'strid'.
  This is used in subsequent calls to FFWRIT and FFCLOS.
```

PK_FFREAD_f_t

```
typedef void (*PK_FFREAD_f_t)(const int *guise, const int *strid,
                              const int *nmax,
                                    char buffer[], int *nactual, int *ifail);

  Read from file

Receives:
  int                     *guise          --- class of file:
                                          ---  FFCROL,FFCSNP,FFCJNL,
                                          ---  FFCXMT,FFCXMO,FFCSCH,FFCLNC
  int                     *strid          --- frustrum strid
  int                     *nmax           --- maximum number of chars to read

Returns:
  char                     buffer[]       --- buffer containing read data
  int                     *nactual        --- actual number of characters read
                                          --- equals *nmax except on EOF/error
  int                     *ifail          --- error code: FR_no_errors
                                          ---             FR_read_fail
                                          ---             FR_end_of_file

Description:
  This function reads from file (starting at the current position of the file
  pointer) storing a maximum of *nmax characters or bytes in the given buffer.
  It returns the actual number of characters or bytes read. This is equal
  to the number requested, except when the end of file is reached or if a read
  error occurs. The end of file error is only returned when no bytes are read.
  The file pointer is incremented by the number of characters or bytes read.
  In the case of a rollback file, Parasolid does not require the function to
  read beyond the 'actsiz' limit which has been returned by FFOPRB.
```

PK_FFSEEK_f_t

```
typedef void (*PK_FFSEEK_f_t)(const int *guise, const int *strid,
                              const int *pos,
                              int *ifail);

  Reset file pointer. This function is documented for legacy
  purposes only, and should not be implemented for new Parasolid applications.

Receives:
  int                     *guise          --- class of file: FFCROL
  int                     *strid          --- frustrum strid
  int                     *pos            --- pointer into file stream
```

```
Returns:
  int                     *ifail          --- error code: FR_no_errors

Description:
  This function resets the file pointer within a rollback file to a value
  noted earlier by FFTELL. Subsequent reading or writing starts from
  this position. Note that the file pointer cannot be set for non-rollback
  files (i.e. those which were opened by FFOPRD or FFOPWR)
```

PK_FFTELL_f_t

```
typedef void (*PK_FFTELL_f_t)(const int *guise, const int *strid,
                              int *pos, int *ifail);

  Output file pointer. This function is documented for legacy
  purposes only, and should not be implemented for new Parasolid applications.

Receives:
  int                     *guise          --- class of file: FFCROL
  int                     *strid          --- frustrum strid

Returns:
  int                     *pos            --- pointer into file stream
  int                     *ifail          --- error code: FR_no_errors

Description:
  This function notes the file pointer within a rollback file at which
  subsequent reading and writing commences. The value of this pointer
  can be restored later by FFSEEK, to allow the same data to be read or
  overwritten. The pointer is set to the beginning of the file when it is
  opened. Note that the pointer cannot be noted for non-rollback files
  (i.e. those which were opened by FFOPRD or FFOPWR).
```

PK_FFWRIT_f_t

```
typedef void (*PK_FFWRIT_f_t)(const int *guise,  const int *strid,
                              const int *nchars, const char buffer[],
                              int *ifail);

  Write to file

Receives:
  int                     *guise          --- class of file:
                                          --- FFCROL,FFCSNP,FFCJNL,
                                          --- FFCXMT,FFCSCH,FFCDBG
  int                     *strid          --- frustrum strid
  int                     *nchars         --- num of chars/bytes to write >=0
  char                     buffer[]       --- buffer containing data to write

Returns:
  int                     *ifail          --- error code: FR_no_errors
                                          ---             FR_write_fail
                                          ---             FR_disc_full

Description:
  This function writes the contents of the buffer to file, starting at the
  current position of the file pointer. The file pointer is incremented by
  the number of characters or bytes written. In the case of a rollback file,
  Parasolid does not require the function to read beyond the 'actsiz'
  limit which was returned by FFOPRB.
```

PK_FGCRCU_f_t

```
typedef void (*PK_FGCRCU_f_t)(const char  *key,
                              int   *keylen,
                              int   *n_kii,
                              int    ki_ints[],
                              int   *n_kir,
                              double ki_reals[],
                              int   *n_data,
                              double fg_data[],
                              int   *ifail);
```

Initialise a foreign curve.

This function is called when an FG curve is requested by the PK function PK_FSURF_create. The purpose of this function is to initialise any FG data that is required by the evaluator and to check that the real and integer data items meet the requirements of the requested evaluator.

A successful call to this function indicates to Parasolid that evaluations of this curve may now be requested.

For more details of this function please refer to the Foreign Geometry documentation.

Specific errors:

FGGEOM	Curve evaluator not available
FGDATA	Curve evaluator data not found
FGFILE	Curve evaluator data allocation fault
FGIERR	Curve evaluator integer data error
FGRERR	Curve evaluator real data error
(FGOPOK	Operation successful)

PK_FGCRSU_f_t

```
typedef void (*PK_FGCRSU_f_t)(const char  *key,
                              int   *keylen,
                              int   *n_kii,
                              int    ki_ints[],
                              int   *n_kir,
                              double ki_reals[],
                              int   *n_data,
                              double fg_data[],
                              int   *ifail);
```

Initialise a foreign surface.

This function is called when an FG surface is requested by the PK function PK_FSURF_create. The purpose of this function is to initialise any FG data that is required by the evaluator. A successful call to this function indicates to Parasolid that evaluations of this surface may now be requested.

Specific errors:

FGGEOM	Surface evaluator not available
FGDATA	Surface evaluator data not found
FGFILE	Surface evaluator data allocation fault
FGIERR	Surface evaluator integer data error
FGRERR	Surface evaluator real data error
(FGOPOK	Operation successful)

PK_FGEVCU_f_t

```
typedef void (*PK_FGEVCU_f_t)(int      ki_ints[],
                              double   ki_reals[],
                              double   fg_data[],
                              double  *t,
                              int     *nderivs,
                              double   results[],
                              int     *ifail);
```

Evaluate a foreign curve.

This function is called when an evaluation of a foreign curve is required by Parasolid. The function should be capable of supplying position and up to second derivatives for a foreign curve which has previously been initialised by a call to FGCRCU.

The argument fg_data identifies the curve evaluator. The arguments ki_ints and ki_reals give the numerical data passed in through Parasolid when the curve was initialised.

For more details of this function please refer to the Foreign Geometry documentation.

Specific errors:

FGEVIN	Evaluation incomplete (not all derivatives requested have been evaluated)
FGOPFA	Evaluation failed
(FGOPOK	Operation successful)

PK_FGEVSU_f_t

```
typedef void (*PK_FGEVSU_f_t)(int      ki_ints[],
                              double   ki_reals[],
                              double   fg_data[],
                              double  *u,
                              double  *v,
                              int     *nu,
                              int     *nv,
                              int     *triang,
                              double   results[],
                              int     *ifail);
```

Evaluate a foreign surface.

This function is called when an evaluation of a foreign surface is required by Parasolid. The function should be capable of supplying position and up to second derivatives for a foreign surface which has previously been initialised by a call to FGCRSU.

The argument fg_data identifies the surface evaluator. The arguments ki_ints and ki_reals give the numerical data passed in through Parasolid at surface initialisation.

The triangular flag, triang, may take the values FGEVSQ (indicating a rectangular array derivatives request) or FGEVTR (indicating a triangular derivatives request). The arguments nu, nv specify the number of u and v derivatives requested.

For more details of this function please refer to the Foreign Geometry documentation.

Specific errors:

FGEVIN	Evaluation incomplete (not all derivatives

```
                              requested have been evaluated)
  FGOPFA                      Evaluation failed
( FGOPOK                      Operation successful )
```

PK_FGPRCU_f_t

```
typedef void (*PK_FGPRCU_f_t)(int      ki_ints[],
                              double   ki_reals[],
                              double   fg_data[],
                              double   range[2],
                              int     *period,
                              int     *ifail);
```

Return foreign curve parameterisation properties.

This function is called, after FGCRCU has initialised the curve, to indicate to Parasolid some of the parametric properties of the curve. The function can specify the parameter range of the curve and indicate whether it is to be treated as periodic.

If the ifail value is set to FGPROP then Parasolid uses default properties for the curve evaluator: there is no need to set the range and periodicity flag. Default properties are parameter range [0, 1] and periodicity flag FGPRBD (i.e. non-periodic parameterisation).

If the ifail value is set to FGOPOK then this function must set the parameter range and periodicity flag:

```
        range[0] = lowest parameter value
        range[1] = highest parameter value
       *period   = FGPRBD (must have non-periodic parameterisation)
```

Note : Closed and periodic curves are NOT permitted for the current version of Parasolid. This function includes the periodic argument for upwards compatibility with future planned enhancements to FG capability.

For more details of this function please refer to the Foreign Geometry documentation.

Specific ifail values:

```
  FGOPFA                      Operation failed
  FGPROP                      Use default properties
( FGOPOK                      Operation successful )
```

PK_FGPRSU_f_t

```
typedef void (*PK_FGPRSU_f_t)(int      ki_ints[],
                              double   ki_reals[],
                              double   fg_data[],
                              double   range[4],
                              int      period[2],
                              int     *ifail);
```

Return foreign surface parameterisation properties.

This function is called, after FGCRSU has initialised the surface, to indicate to Parasolid some of the parametric properties of the surface. The function can specify the parameter ranges of the surface and indicate whether it is to be treated as periodic in u and/or v.

```
If the ifail value is set to FGPROP then Parasolid uses default
properties for the surface evaluator: there is no need to set the range and
periodicity flag. Default properties are parameter ranges [0, 1] for both
u and v parameters and periodicity flags FGPRBD for both parameters
(i.e. non-periodic parameterisation).

If the ifail value is set to FGOPOK then this function must set the range
array and the periodicity flags array.

     range[0], range[1] are lower and upper u parameter values
     range[2], range[3] are lower and upper v parameter values
     period[0] is the u parameter periodicity flag (must be FGPRBD)
     period[0] is the v parameter periodicity flag (must be FGPRBD)

Note : Closed and periodic surfaces are NOT permitted in the current
       version of Parasolid. This function includes the periodic argument for
       upwards compatibility with future planned enhancements to FG
       capability.

Specific ifail values:
  FGOPFA              Operation failed
  FGPROP              Use default parameter properties
( FGOPOK              Operation successful )
```

PK_field_names_t

```
typedef union PK_field_names_u
    {
    char       **names;
    PK_UCHAR_t **unames;
    }
    PK_field_names_t;
```

This union describes an array of either native-character strings, or Unicode strings, used to specify the names of fields in an attribute definition.

PK_fill_hole_fault_t

```
typedef int PK_fill_hole_fault_t;
```

This data type is used to return fault types associated with filling a hole. In some cases faulty topological entities are also returned. It has the following values:

	fault_topols PK_CLASS_...	
PK_fill_hole_ok_c		succeeded without any problems
PK_fill_hole_bad_edge_c	edge	an edge has been supplied which is not on the boundary of a hole
PK_fill_hole_duplicate_c	edge	duplicate edges supplied
PK_fill_hole_vertex_c	vertex	more than two edges meet at vertex geometry
PK_fill_hole_too_many_loops_c		more than one loop of edge geometry
PK_fill_hole_gap_c		gap in supplied boundary edges
PK_fill_hole_no_edge_on_target_c		none of the supplied edges is an edge of the target body
PK_fill_hole_cant_match_body_c	body	can't match any edges with a body
PK_fill_hole_cant_match_edge_c	edge	can't match edge or portion of edge with a face
PK_fill_hole_too_small_c	edge	the fill sheet body is too small to cover the hole

```
    PK_fill_hole_unknown_c                  internal unspecified failure
    PK_fill_hole_face_c             face(s) succeeded, but invalid face(s)
                                            have been produced
    PK_fill_hole_face_face_c        faces   succeeded, but a face-face
                                            inconsistency has been produced
    PK_fill_hole_not_smooth_c       vertex  boundary of hole is not smooth
    PK_fill_hole_too_complex_c              no suitable fill method exists
```

PK_fill_hole_method_t

typedef int PK_fill_hole_method_t;

This data type specifies the method which will be used to fill a hole.

It has the values :

```
    PK_fill_hole_trim_to_hole_c   patch hole by first trimming supplied fill
                                  sheet body to the boundary of the hole
    PK_fill_hole_trim_to_sheet_c  patch hole by first trimming target bodies to
                                  the laminar edges of supplied fill sheet body
    PK_fill_hole_create_patch_c   patch hole by creating a suitable fill sheet
                                  body to match the boundary of the hole
```

PK_FIN_array_t

This structure contains an array of entities of type PK_FIN_t

```
typedef struct PK_FIN_array_s
    {
    PK_FIN_t       *array;   --- array of FINs
    int             length;  --- length of array
    }
    PK_FIN_array_t;
```

It has the fields:

'array': array of entities of type PK_FIN_t

'length': the number of entities in 'array'

PK_FIN_t

This type represents a FIN. A fin is the junction between an edge and a face.

A fin is a type of topology (PK_CLASS_fin is a subclass of PK_CLASS_topol).

PK_FIN_type_t

typedef int PK_FIN_type_t;

This datatype contains classification data for a fin.

Value can be one of:

```
    PK_FIN_type_wire_c          The fin's owning edge has another fin
                                in the same face, and both fins are in the
                                same loop.
    PK_FIN_type_biwire_c        The fin's owning edge has another fin
```

```
                                in the same face, but the two fins are in
                                different loops.
    PK_FIN_type_normal_c        The fin's owning edge has no other fin
                                in the same face.
```

PK_FMALLO_f_t

typedef void (*PK_FMALLO_f_t)(int *nbytes, char **memory, int *ifail);

```
  Allocate virtual memory

Receives:
  int                    *nbytes      --- length of memory region in bytes

Returns:
  char                  **memory      --- pointer to start of memory region
  int                    *ifail       --- error code: FR_no_errors
                                      ---             FR_memory_full

Description:
  This function allocates the specified amount of virtual memory (in bytes),
  returning a pointer to the start address. The memory that is allocated
  can be accessed using byte addresses in the range memory[0] to
  memory[nbytes-1].

  Parasolid assumes that the block of virtual memory can be filled with data
  of any type, starting from the byte address which is returned. On machines
  with alignment restrictions, the start address of a block of space must be
  aligned onto an appropriate word boundary.

  Space allocation is likely to be in the order of between 0.1 and 10 M bytes.
  If there is insufficient space to satisfy a request, none is allocated.
```

PK_FMFREE_f_t

typedef void (*PK_FMFREE_f_t)(int *nbytes, char **memory, int *ifail);

```
  Free virtual memory

Receives:
  int                    *nbytes      --- length of memory region in bytes
  char                  **memory      --- pointer to start of memory region

Returns:
  int                    *ifail       --- error code: FR_no_errors

Description:
  Any space which has been allocated by calling FMALLO must be freed by a call
  to FMFREE, passing it the start address of the block of virtual memory and
  size of memory which was requested. The space must be freed by a single call.
```

PK_FSTART_f_t

typedef void (*PK_FSTART_f_t)(int *ifail);

```
  Start up the frustrum

Returns:
  int                    *ifail          --- error code: FR_no_errors

Description:
  This function initialises the frustrum. It is called from kernel
```

```
  function PK_SESSION_start and is the first frustrum function to be called.
  Calls to FSTART and FSTOP (which closes down the Frustrum) may be
  nested, in which case only the "outermost calls" (i.e. the first FSTART
  call, and the "balancing" FSTOP call) should have any effect. However,
  once the frustrum has been closed down, a further FSTART call should
  cause it to be re-initialised.
```

PK_FSTOP_f_t

```
typedef void (*PK_FSTOP_f_t)(int *ifail);

  Shut down the frustrum

Returns:
  int                      *ifail          --- error code: FR_no_errors

Description:
  This function shuts down the frustrum, and is called from kernel
  function PK_SESSION_stop. Calls to FSTART and FSTOP may be nested;
  in this case only the "outermost calls" (i.e. the first FSTART call,
  and the "balancing" FSTOP call) should have any effect.
```

PK_FSURF_array_t

This structure contains an array of entities of type PK_FSURF_t

```
typedef struct PK_FSURF_array_s
    {
    PK_FSURF_t     *array;   --- array of Foreign SURFaces
    int             length;  --- length of array
    }
    PK_FSURF_array_t;

It has the fields:

'array':   array of entities of type PK_FSURF_t

'length':  the number of entities in 'array'
```

PK_FSURF_sf_t

```
    struct PK_FSURF_sf_s
        {
        char       *key;       --- surf key
        int         n_ints;    --- number of integer values (>=0)
        int        *ints;      --- integer values
        int         n_doubles; --- number of double values  (>=0)
        double     *doubles;   --- double values
        size_t      space;     --- data space required      (>=0)
        PK_TRANSF_t transf;    --- transformation (may be PK_ENTITY_null)
        };
    typedef struct PK_FSURF_sf_s PK_FSURF_sf_t;
```

This structure is the standard form of a foreign surface.

Specific Errors

```
PK_ERROR_FG_eval_not_found      surface evaluator not found
PK_ERROR_FG_data_alloc_error    surface evaluator data allocation fault
PK_ERROR_FG_doubles             'doubles' data error
PK_ERROR_FG_ints                'ints' data error
PK_ERROR_FG_data_not_found      surface evaluator data not found
PK_ERROR_FG_evaluator_error     surface evaluator failure
PK_ERROR_FG_modelling_error     cannot model with this surface
```

A separate Foreign Geometry User Guide is available.

Sufficient data space is always allocated to store one double. The 'space' field indicates data space in addition to this.

PK_FSURF_t

This type represents a Foreign SURFace; that is, one implemented in an externally supplied parametric evaluator.

An fsurf is a type of surface (PK_CLASS_fsurf is a subclass of PK_CLASS_surf).

The fsurf is defined by data contained in a PK_FSURF_sf_t.

PK_fxf_error_t

```
struct PK_fxf_error_s
    {
    PK_fxf_fault_t        fault;        --- blend fault
    int                   n_topols;     --- number of topols
    PK_TOPOL_t           *topols;       --- associated topology
    int                   n_points;     --- number of points
    PK_VECTOR_t          *points;       --- error points
    int                   n_dists;      --- number of lengths
    double               *dists;        --- error lengths
    };
typedef struct PK_fxf_error_s PK_fxf_error_t;
```

This structure returns blending faults and associated data

This typedef is described in the documentation of PK_FACE_make_blend.

PK_fxf_fault_t

```
typedef int PK_fxf_fault_t;
```

This data type is described in the documentation of PK_FACE_make_blend.

PK_GEOM_array_t

This structure contains an array of entities of type PK_GEOM_t

```
typedef struct PK_GEOM_array_s
    {
    PK_GEOM_t      *array;   --- array of GEOMetrical entities
```

```
    int             length;  --- length of array
    }
    PK_GEOM_array_t;
```

It has the fields:

'array': array of entities of type PK_GEOM_t

'length': the number of entities in 'array'

PK_GEOM_check_o_t

```
struct PK_GEOM_check_o_s
    {
    int              o_t_version;    --- version number of option structure
    int              max_faults;     --- limit on fault numbers (1)
    PK_check_geom_t  geom;           --- bad geometry (PK_check_geom_yes_c)
    };

typedef struct PK_GEOM_check_o_s PK_GEOM_check_o_t;
```

Holds optional controls for checking a geometrical entity.

The option structure defines options applicable to checking a geometrical entity.

PK_GEOM_range_array_o_t

```
struct PK_GEOM_range_array_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    PK_range_bound_t    bound;          --- bounds on min separation
    };

typedef struct PK_GEOM_range_array_o_s PK_GEOM_range_array_o_t;
```

Holds optional controls for finding the global minimum separations between two arrays of geometrical entities.

The option structure defines options applicable to finding the global minimum separations between two arrays of geometrical entities.

PK_GEOM_range_array_vector_o_t

```
struct PK_GEOM_range_array_vector_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    PK_range_bound_t    bound;          --- bounds on min separation
    };

typedef struct PK_GEOM_range_array_vector_o_s PK_GEOM_range_array_vector_o_t;
```

Holds optional controls for finding the global minimum separations between an array of geometrical entities and a position.

The option structure defines options applicable to finding the global minimum separations between an array of geometrical entities and a position.

PK_GEOM_range_local_o_t

```
struct PK_GEOM_range_local_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    };

typedef struct PK_GEOM_range_local_o_s PK_GEOM_range_local_o_t;
```

Holds optional controls for finding the local minimum separations between two geometrical entities.

The option structure defines options applicable to finding the local minimum separations between two geometrical entities.

PK_GEOM_range_local_vector_o_t

```
struct PK_GEOM_range_local_vector_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    };

typedef struct PK_GEOM_range_local_vector_o_s PK_GEOM_range_local_vector_o_t;
```

Holds optional controls for finding the local minimum separations between a geometrical entity and a position.

The option structure defines options applicable to finding the local minimum separations between a geometrical entity and a position.

PK_GEOM_range_o_t

```
struct PK_GEOM_range_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    PK_range_bound_t    bound;          --- bounds on min separation
    PK_range_guess_t    guesses[2];     --- guesses for each end
    };

typedef struct PK_GEOM_range_o_s PK_GEOM_range_o_t;
```

Holds optional controls for finding the global minimum separation between two geometrical entities.

The option structure defines options applicable to finding the global minimum separation between two geometrical entities.

PK_GEOM_range_vector_o_t

```
struct PK_GEOM_range_vector_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    PK_range_bound_t    bound;          --- bounds on min separation
    PK_range_guess_t    guess;          --- guess for end on 'geom'
    };

typedef struct PK_GEOM_range_vector_o_s PK_GEOM_range_vector_o_t;
```

Holds optional controls for finding the global minimum separation between a geometrical entity and a position.

The option structure defines options applicable to finding the global minimum separation between a geometrical entity and a position.

PK_GEOM_render_line_o_t

```
struct PK_GEOM_render_line_o_s
    {
    int                     o_t_version;  --- version of this PK option struct
    PK_render_param_t       param;        --- whether to draw parametric hatch
                                          --- lines (PK_render_param_no_c)
    PK_render_boundary_t    boundary;     --- whether to draw surface boundaries
                                          --- (PK_render_boundary_yes_c)
    PK_render_bcurve_t      bcurve;       --- method for rendering bcurves
                                          --- (PK_render_bcurve_polyline_c)
    PK_LOGICAL_t              is_curve_chord_tol;
                                          --- whether curve_chord_tol is set
                                          --- (PK_LOGICAL_false)
    double                    curve_chord_tol;
                                          --- chordal tolerance between curve
                                          --- and chord edges in model units
                                          --- (0.0)
    PK_LOGICAL_t              is_curve_chord_max;
                                          --- whether curve_chord_max is set
                                          --- (PK_LOGICAL_false)
    double                    curve_chord_max;
                                          --- max length of chord in model units
                                          --- (0.0)
    PK_LOGICAL_t              is_curve_chord_ang;
                                          --- whether curve_chord_ang is set
                                          --- (PK_LOGICAL_false)
    double                    curve_chord_ang;
                                          --- max angle between adjacent chords
                                          --- in radians (0.0)
    double                  param_u;      --- hatch spacing in U direction (0.0)
    double                  param_v;      --- hatch spacing in V direction (0.0)
    };
typedef struct PK_GEOM_render_line_o_s PK_GEOM_render_line_o_t;
```

Controls for the PK_GEOM_render_line options structure.

Specific Errors

PK_ERROR_o_t_version_incorrect	option structure not initialised correctly
PK_ERROR_bad_tolerance	is_<tol> is true but used with a bad <tol>
PK_ERROR_bad_parameter	PK_render_param_yes_c needs param_u/v vals

The following PK option structure provides controls for PK_GEOM_render_line when generating view-independent drawings of geometric entities through the Graphics Output (GO) interface.

The options specify whether surface boundary and hatch lines are required and the method to be used when outputting bcurves through GO.

tolerance values
Each control tolerance is passed as a pair of values, consisting of a PK_LOGICAL_t flag and a double precision value. If a specific tolerance is required, the variable is set to a strictly positive value and the associated flag set to PK_LOGICAL_true.

If Parasolid is to use its internal tolerance defaults, the associated flag is set to PK_LOGICAL_false (and no use made of the associated tolerance argument). This is the default state set by the PK_GEOM_render_line_o_m macro.

This convention has changed since Parasolid 8.0. Control tolerances were formerly passed as doubles, using the convention that a value of exactly 0.0 implied that Parasolid should choose a suitable tolerance value.

PK_GEOM_t

This type represents a GEOMetrical item.

A geometrical item is a type of entity (PK_CLASS_geom is a subclass of PK_CLASS_entity).

A geometrical item is of one of the types:

- PK_SURF_t: a surface
- PK_CURVE_t: a curve
- PK_POINT_t: a point

PK_GOCLPX_f_t

```
typedef void (*PK_GOCLPX_f_t)(const int *nreals, const double *rvals,
                              const int *nints,  const int *ivals,
                                    int *ifail);
```

Close output of encoded pixel data. This function is documented for legacy purposes only, and should not be implemented for new Parasolid applications.

The received arguments (nreals, rvals, ninit, ivals) should be ignored.

ifail

The returned argument 'ifail' is a code indicating whether the Frustrum wants to abort graphic output; it must be set to one of the values

```
CONTIN = 0;      --- Continue: no errors
ABORT  = -1011;  --- Abort: return control to caller
```

PK_GOCLSG_f_t

```
typedef void (*PK_GOCLSG_f_t)(const int *segtyp,
                              const int *ntags, const int    *tags,
                              const int *ngeom, const double *geom,
                              const int *nlntp, const int    *lntp,
                                    int *ifail);
```

Close hierarchical segment.

The arguments have the following significance :-

segtyp: type of segment

'segtyp' - the type of the segment; one of the following values

```
SGTPBY = 2003;      --- body
SGTPED = 2006;      --- edge
SGTPSI = 2007;      --- silhouette
SGTPPH = 2008;      --- planar hatch-line
SGTPRH = 2009;      --- radial hatch-line
SGTPPL = 2012;      --- parametric hatch-line
SGTPFA = 2017;      --- face
SGTPGC = 2019;      --- geometry ... curve
SGTPGS = 2020;      --- geometry ... surface
```

ntags, tags: tags associated with segment

'ntags', 'tags' - an array of tags associated with the segment.
The tags given depend upon the segment type as follows:

```
SGTPBY                        --- tag of body
SGTPED                        --- tag of edge
SGTPSI/SGTPPH/SGTPRH/SGTPPL   --- tag of face
SGTPFA                        --- tag of face
SGTPGC                        --- tag of curve
SGTPGS                        --- tag of surface
```

ngeom, geom: geometry of segment

'ngeom', 'geom' - an array of reals giving the geometry of the segment.
The geometry given depends upon the segment type as follows:

```
SGTPBY
    - model space box of the body: given as two vectors:
      (xmin, ymin, zmin), (xmax, ymax, zmax).
SGTPED/SGTPSI/SGTPPH/SGTPRH/SGTPPL
    - no geometry returned
SGTPFA
    - model space box of the face: given same as a body box.
SGTPGC
    - model space box of the curve.
SGTPGS
    - model space box of the surface.
```

nlntp, lntp: type of geometry and other codes

'nlntp', 'lntp' - an array of integers.
lntp[0] - Occurrence number of the entity from which the segment was derived.
lntp[1] - silhouette label if segtyp is SGTPSI

ifail:

```
    'ifail' - a code indicating whether the Frustrum wants
              to abort graphic output.

              One of the values:

              CONTIN = 0;      --- Continue: no errors
              ABORT  = -1011;  --- Abort: return control to caller
```

PK_GOOPPX_f_t

```
typedef void (*PK_GOOPPX_f_t)(const int *nreals, const double *rvals,
                              const int *nints,  const int    *ivals,
                                    int *ifail);
```

Open output of encoded pixel data. This function is documented for legacy purposes only, and should not be implemented for new Parasolid applications.

```
 Opens the output of pixel data.

 The arguments have the following significance :-

    nreals and rvals
    'nreals'     - length of the real array = 4

    'rvals[0]' } - image space position of top left of picture = (Xmin, Ymin)
    'rvals[1]' }

    'rvals[2]' } - image space position of bottom right of picture =
    'rvals[3]' }                                                    (Xmax, Ymax)

    ninits and ivals
    'nints'      - length of the integer array = 4

    'ivals[0]' } - pixel at top left of picture = (xmin, ymin)
    'ivals[1]' }

    'ivals[2]' } - pixel at bottom right of picture = (xmax, ymax)
    'ivals[3]' }

    ifail

    'ifail' - a code indicating whether the Frustrum wants
              to abort graphic output; one of the values
              CONTIN = 0;      --- Continue: no errors
              ABORT  = -1011;  --- Abort: return control to caller
```

PK_GOOPSG_f_t

```
typedef void (*PK_GOOPSG_f_t)(const int *segtyp,
                              const int *ntags, const int    *tags,
                              const int *ngeom, const double *geom,
                              const int *nlntp, const int    *lntp,
                                    int *ifail);
```

Open hierarchical segment.

```
The arguments have the following significance :-

    segtyp: type of segment
```

```
'segtyp' - the type of the segment; one of the following values

    SGTPBY = 2003;      --- body
    SGTPED = 2006;      --- edge
    SGTPSI = 2007;      --- silhouette
    SGTPPH = 2008;      --- planar hatch-line
    SGTPRH = 2009;      --- radial hatch-line
    SGTPPL = 2012;      --- parametric hatch-line
    SGTPFA = 2017;      --- face
    SGTPGC = 2019;      --- geometry ... curve
    SGTPGS = 2020;      --- geometry ... surface


ntags, tags: tags associated with segment

'ntags', 'tags' - an array of tags associated with the segment.
    The tags given depend upon the segment type as follows:

      SGTPBY                        --- tag of body
      SGTPED                        --- tag of edge
      SGTPSI/SGTPPH/SGTPRH/SGTPPL   --- tag of face
      SGTPFA                        --- tag of face
      SGTPGC                        --- tag of curve
      SGTPGS                        --- tag of surface


ngeom, geom: geometry of segment

'ngeom', 'geom' - an array of reals giving the geometry of the segment.
    The geometry given depends upon the segment type as follows:

        SGTPBY
            - model space box of the body: given as two vectors:
              (xmin, ymin, zmin), (xmax, ymax, zmax).
        SGTPED/SGTPSI/SGTPPH/SGTPRH/SGTPPL
            - no geometry returned
        SGTPFA
            - model space box of the face: given same as a body box.
        SGTPGC
            - model space box of the curve
        SGTPGS
            - model space of the surface


nlntp, lntp: type of geometry and other codes

'nlntp', 'lntp' - an array of integers.
    lntp[0] - Occurrence number of the entity from which the segment
              was derived.
    lntp[1] - Silhouette label if segtyp is SGTPSI


ifail:

'ifail' - a code indicating whether the Frustrum wants
          to abort graphic output.

          One of the values:

          CONTIN = 0;      --- Continue: no errors
          ABORT  = -1011;  --- Abort: return control to caller
```

PK_GOPIXL_f_t

```
typedef void (*PK_GOPIXL_f_t)(const int *npixels, const double *rvals,
                              const int *nints,   const int    *ivals,
                                    int *ifail);
```

Output encoded pixel data. This function is documented for legacy purposes only, and should not be implemented for new Parasolid applications.

```
The arguments have the following significance :-

   npixels and rvals
   'npixels' is the number of pixels to output in a row (ie. scan line)

   'rvals' contains 3 * 'npixels' reals. Each group consists of
   3 positive real values :-

       1.  Intensity of red for the pixel

       2.  Intensity of green for the pixel

       3.  Intensity of blue for the pixel

   nints and ivals
   'nints' and 'ivals' should be ignored.

   ifail

   'ifail' - a code indicating whether the Frustrum wants
             to abort graphic output; one of the values
             CONTIN = 0;      --- Continue: no errors
             ABORT  = -1011;  --- Abort: return control to caller
```

PK_GOSGMT_f_t

```
typedef void (*PK_GOSGMT_f_t)(const int *segtyp,
                              const int *ntags, const int    *tags,
                              const int *ngeom, const double *geom,
                              const int *nlntp, const int    *lntp,
                                    int *ifail);
```

Output non hierarchical segment

```
The arguments have the following significance :-

    segtyp: type of segment

    'segtyp' - the type of the segment; one of the following values

        SGTPED = 2006;   --- Edge
        SGTPSI = 2007;   --- Silhouette line
        SGTPPH = 2008;   --- Planar hatch-line
        SGTPRH = 2009;   --- Radial hatch-line
        SGTPRU = 2010;   --- Rib line (unfixed blend)
        SGTPBB = 2011;   --- Blend-boundary line
        SGTPPL = 2012;   --- Parametric hatch line

        debugging segment types
        SGTPCU = 2014;   --- Curve line *** Used only for debugging ***
        SGTPBX = 2015;   --- Box        *** Used only for debugging ***
        SGTPFT = 2016;   --- Facet
        SGTPER = 2018;   --- Error segment
```

```
        SGTPGC = 2019;   --- Geometry ... curve.
        SGTPGS = 2020;   --- Geometry ... surface.
        SGTPGB = 2021;   --- Geometry ... surface boundary.
        SGTPMF = 2022;   --- Mangled facet
        SGTPVT = 2023;   --- Visibility segment (used for hierarchical output)
        SGTPTS = 2024;   --- Facet strip
        SGTPVP = 2025;   --- Parametrised Visibility segment


        When rendering a list of entities Parasolid may encounter a body, face
        or edge that it is unable to render (e.g. a rubber face). In such
        cases, Parasolid outputs an error segment (SGTPER), giving the tag
        of the bad entity and a code indicating why it was unable to render
        it. Parasolid then continues to render the remaining entities.
        If, during a call to PK_TOPOL_render_facet, either

          *  user tolerances can't be matched, or
          *  facets are created which self intersect or are severely creased,

        then geometric data is output as a segment type SGTPMF. In this case,
        the facets are always triangular.
        This does not apply to geometry (i.e. SGTPGC, SGTPGS and SGTPGB). These
        are never replaced by error segments, unrenderable geometry
        segments are not output at all.
        If edges/silhouettes/hatchlines are being output hierarchically then
        GOSGMT is called to output the whole geometry of the item
        (optional) and again to output the positions at which the visibility
        changes on the item (a visibility segment).


    ntags, tags: tags associated with segment

    'ntags', 'tags' - an array of tags associated with the segment.
        The tags given depend upon the segment type as follows:
            SGTPED
                - tag of edge (or none if edge output hierarchically)
            SGTPSI, SGTPPH, SGTPRH
                - tag of face on which silhouette or hatch line lies (or none
                if silhouette or hatchline output hierarchically)
            SGTPPL
                - tag of face or surface on which the parametric hatch line
                  lies (or none if hatchline output hierarchically)
            SGTPRU, SGTPBB
                - tag of edge blended
            SGTPFT
                - tag of face on which facet lies
            SGTPMF
                - tag of face on which mangled facet lies
            SGTPTS
                - tag of face on which facet strip lies
            SGTPGC
                - tag of the curve
            SGTPGS
                - tag of the surface
            SGTPGB
                - tag of the surface for which this is a boundary curve
            SGTPVT
                - none
            SGTPER
                - tag of entity which could not be rendered
            SGTPVP
                - none

        If edges, silhouettes or hatchlines are output hierarchically then
        the tag information is given by GOOPSG and GOCLSG.

        If regional data was requested in a call to PK_TOPOL_render_line two
```

```
        further tags are given for SGTPED and SGTPSI segments, identifying
        the faces either side of the line in the image. Either or both face
        tags may be null.

        If edge tags were requested in a call to PK_TOPOL_render_facet, for
        each edge of the facet the tag of the model edge from which it was
        derived is given; or a null tag if it is not derived from a model
        edge. The number of edge tags given equals the number of vertices
        given in 'geom'. The first edge tag ( tags[1] ) is the tag of
        the edge from which is derived the first facet edge (which ends at
        the first vertex given in geom). The second edge tag is for
        the facet edge which ends at the second vertex, and so on.

        If facet strips have been requested in a call to PK_TOPOL_render_facet
        the format for the edges tags is as follows:-

             If the strip has n vertices, then there are

                2 * (n - 2) + 1 edge tags

             The i-th element of the tags array (starting at i = 1) is
             the tag of the edge between vertex,

                floor ( (i - 1) / 2 ) + 1

            and vertex,

                floor (i / 2) + 2

            where, floor (x) is the integer portion of x.


    ngeom, geom: geometry of segment

    'ngeom', 'geom' - an array of reals giving the geometry of the segment.
        The values given depend upon the type of the geometry, as specified in
        the second element of 'lntp'.

            L3TPSL - straight line: ngeom = 9
                start point, end point, line direction.

            L3TPCC - complete circle: ngeom = 7
                centre point, axis direction, radius.

                The forward direction of the circle is clockwise when viewing
                the circle along the axis direction.

            L3TPCI - circular arc: ngeom = 13
                centre point, axis direction, radius, start point, end point.

                The forward direction of the circle is given above.

            L3TPCE - complete ellipse: ngeom = 11
                centre point, major axis direction, minor axis direction,
                major radius, minor radius

                The forward direction of the ellipse is clockwise when viewing
                the ellipse along the axis direction. The axis direction is
                the vector cross product of the major axis with the minor axis
                in that order.

            L3TPEL - elliptical arc: ngeom = 17
                centre point, major axis direction, minor axis direction,
                major radius, minor radius, start point, end point

                The forward direction of the ellipse is given above.
```

```
L3TPPY - poly line:
    'geom' holds 'ngeom' vectors.

L3TPPC - non-rational B-curve in Bezier form:
    'geom' holds 'ngeom' vectors of dimension 3. These are the
    Bezier vertices of the curve.

L3TPRC - rational B-curve in Bezier form:
    'geom' holds 'ngeom' vectors of dimension 4. These are the
    Bezier vertices of the curve, where each Bezier vertex
    consists of a 3-space point and a weight.

L3TPNC - non-rational B-curve in NURBs form:
    'geom' holds 'ngeom' reals. These consist of lntp[9] vectors
    of dimension 3, which are the bspline vertices followed by
    lntp[10] reals which are the knots. So ngeom is equal to
    3 times lntp[9] plus lntp[10].

L3TPRN - rational B-curve in NURBs form:
    'geom' holds 'ngeom' reals. These consist of lntp[9] vectors
    of dimension 4, which are the bspline vertices where each
    vertex consists of a 3-space point followed by a weight. These
    are followed by lntp[10] reals which are the knots. So ngeom
    is equal to 4 times lntp[9] plus lntp[10].

L3TPFV - facet vertices:
    'geom' holds 'ngeom' vectors defining the vertices of
    a facet.

L3TPFN - facet vertices plus surface normals:
    'geom' holds
      * 'ngeom'/2 vectors defining the vertices of a
        facet followed by
      * 'ngeom'/2 vectors defining the surface
        normals at the vertices.

L3TPFP - facet vertices plus parameters
    'geom' holds
      * 'ngeom'/2 vectors defining the vertices of a
        facet followed by
      * 'ngeom'/2 vectors defining the surface
        and curve parameters (u,v,t) of the vertex

L3TPFI - facet vertices plus normals plus parameters
    'geom' holds
      * 'ngeom'/3 vectors defining the vertices of a
         facet followed by
      *  'ngeom'/3 vectors defining the surface
         normals at the vertices followed by
      *  'ngeom'/3 vectors defining the surface and curve
         parameters (u,v,t) of the vertex

L3TPF1 - facet vertices + normals + parameters + 1st derivs
    'geom' holds
      * 'ngeom'/5 vectors defining the vertices of a
        facet followed by
      * 'ngeom'/5 vectors defining the surface normals at the
         vertices followed by
      * 'ngeom'/5 vectors defining the surface and curve
         parameters (u,v,t) of the vertex followed by
      * 'ngeom'/5 vectors defining the dP/du surface
        derivatives at the vertices followed by
      * 'ngeom'/5 vectors defining the dP/dv surface
        derivatives at the vertices

L3TPF2 - facet vertices + normals + parameters + all derivs
    'geom' holds
```

```
      * 'ngeom'/8 vectors defining the vertices of a
        facet followed by
      * 'ngeom'/8 vectors defining the surface normals at the
         vertices followed by
      * 'ngeom'/8 vectors defining the surface and curve
         parameters (u,v,t) of the vertex followed by
      * 'ngeom'/8 vectors defining the dP/du surface
        derivatives at the vertices followed by
      * 'ngeom'/8 vectors defining the dP/dv surface
        derivatives at the vertices followed by
      * 'ngeom'/8 vectors defining the d2P/du2 surface
        derivatives at the vertices followed by
      * 'ngeom'/8 vectors defining the d2P/dudv surface
        derivatives at the vertices followed by
      * 'ngeom'/8 vectors defining the d2P/dv2 surface
        derivatives at the vertices

L3TPTS - facet strip vertices:
    'geom' holds 'ngeom' vectors defining the vertices of
    a facet strip.

L3TPTN - facet strip vertices plus surface normals:
    'geom' holds
      * 'ngeom'/2 vectors defining the vertices of a
        facet strip followed by
      * 'ngeom'/2 vectors defining the surface normals
        at the vertices.

L3TPTN - facet strip vertices plus parameters:
    'geom' holds
      * 'ngeom'/2 vectors defining the vertices of a
        facet strip followed by
      * 'ngeom'/2 vectors defining the surface and curve
        parameters (u,v,t) of the vertex

L3TPTI - facet strip vertices plus normals plus parameters:
    'geom' holds
      * 'ngeom'/3 vectors defining the vertices of a
        facet strip followed by
      * 'ngeom'/3 vectors defining the surface normals
        followed by
      * 'ngeom'/3 vectors defining the parameters of the vertex

L3TPT1 - facet strip vertices + normals + parameters + 1st derivs
    'geom' holds
      * 'ngeom'/5 vectors defining the vertices of a
        facet strip followed by
      * 'ngeom'/5 vectors defining the surface normals at the
        vertices followed by
      * 'ngeom'/5 vectors defining the surface and curve
        parameters (u,v,t) of the vertex followed by
      * 'ngeom'/5 vectors defining the dP/du surface derivatives
        at the vertices followed by
      * 'ngeom'/5 vectors defining the dP/dv surface derivatives
        at the vertices

L3TPT2 - facet strip vertices + normals + parameters + all derivs
    'geom' holds
      * 'ngeom'/8 vectors defining the vertices of a
        facet strip followed by
      * 'ngeom'/8 vectors defining the surface normals at the
        vertices followed by
      * 'ngeom'/8 vectors defining the surface and curve
        parameters (u,v,t) of the vertex followed by
      * 'ngeom'/8 vectors defining the dP/du surface derivatives
        at the vertices followed by
      * 'ngeom'/8 vectors defining the dP/dv surface derivatives
```

```
                  at the vertices followed by
                * 'ngeom'/8 vectors defining the d2P/du2 surface derivatives
                  at the vertices followed by
                * 'ngeom'/8 vectors defining the d2P/dudv surface
                  derivatives at the vertices followed by
                * 'ngeom'/8 vectors defining the d2P/dv2 surface
                  derivatives at the vertices
```

If a facet has multiple loops the outer loop is output first and the inner loops follow. The vertices of the outer loop are ordered anticlockwise when viewed down the surface normal. The vertices of inner loops are ordered clockwise. Facets are manifold; i.e. no vertex coincides with any other in the same facet, nor does it lie in any edge in the same facet.

For facet strips the vertices of the first facet in the strip are held in elements 0,1 and 2 of 'geom' and are ordered anticlockwise when viewed down the surface normal. The vertices of the second facet are held in elements 1,2 and 3 and are ordered clockwise. The vertices of the n-th facet are held in elements n-1, n and n+1 and are ordered anticlockwise if n is odd and clockwise if n is even. If lntp[1] is L3TPTS then the number of facets in a strip is given by 'ngeom' - 2. If lntp[1] is L3TPTN then the number of facet is given by 'ngeom'/2 - 2.

For visibility segments (SGTPVT/SGTPVP) geom contains the visibility transition points (i.e. vectors in model space at which the edge changes visibility). Segments of type SGTPVP contain parameters as well. If the edge has only one visibility then no geometry is output.

The geometry array in an unparametrised (SGTPVT) visibility segment holds a sequence of sets of three doubles {x,y,z} corresponding to the vector position of the transition points in model space.

The geometry array in a parametrised (SGTPVP) visibility segment holds a sequence of sets of four doubles {x,y,z,t} corresponding to the vector position of the transition points and their parameter on the geometry segment.

For error segments (SGTPER) 'ngeom' is zero.

nlntp, lntp: type of geometry and other codes

'nlntp', 'lntp' - an array of integers specifying the type of geometry of the segment and other codes as follows:

lntp[0] - Occurrence number of the entity from which the segment was derived.

If the segment type is SGTPER

lntp[1] - Reason why Parasolid is unable to render the entity:

```
ERNOGO = 4001;   --- Unspecified error
ERRUBB = 4002;   --- Rubber entity (no geometry attached)
ERSANG = 4003;   --- Surface angular tolerance too small
ERSDIS = 4004;   --- Surface distance tolerance too small
ERCANG = 4005;   --- Curve angular tolerance too small
ERCDIS = 4006;   --- Curve distance tolerance too small
ERCLEN = 4007;   --- Chord chord length tolerance too small
ERFWID = 4008;   --- Facet width tolerance too small
```

If the segment type is SGTPVT or SGTPVP

* lntp contains an array of visibility codes for the edge/silhouette/hatchline.
* lntp+nlntp holds the smoothness codes for the edge

```
    See the Parasolid Downward Interfaces manual for more information.

Otherwise

    lntp[1] - Geometry type; one of the values:
            L3TPSL = 3001;   --- Straight line
            L3TPCI = 3002;   --- Partial circle
            L3TPCC = 3003;   --- Complete circle
            L3TPEL = 3004;   --- Partial ellipse
            L3TPCE = 3005;   --- Complete ellipse
            L3TPPY = 3006;   --- Poly-line
            L3TPFV = 3007;   --- Facet vertices
            L3TPFN = 3008;   --- Facet vertices plus surface normals
            L3TPPC = 3009;   --- Non-rational B-curve (Bezier)
            L3TPRC = 3010;   --- Rational B-curve (Bezier)
            L3TPTS = 3011;   --- Facet strip vertices
            L3TPTN = 3012;   --- Facet strip vertices plus surface normals
            L3TPNC = 3013;   --- Non-rational B-curve (NURBs form)
            L3TPRN = 3014;   --- Rational B-curve (NURBs form)
            L3TPFP = 3015;   --- Facet vertices plus parameters
            L3TPFI = 3016;   --- Facet vertices plus norms plus parameters
            L3TPTP = 3017;   --- Facet strip vertices plus parameters
            L3TPTI = 3018;   --- Facet strip verts plus norms plus params
            L3TPF1 = 3019;   --- Facet verts + norms + params + 1st derivs
            L3TPF2 = 3020;   --- Facet verts + norms + params + all derivs
            L3TPT1 = 3021;   --- Facet strip verts+norms+params+1st derivs
            L3TPT2 = 3022;   --- Facet strip verts+norms+params+all derivs

If the geometry type is anything except L3TPF* or L3TPT*:

    lntp[2] - Completeness; one of the values
            CODCOM = 1001;   --- Segment complete
            CODINC = 1002;   --- Segment incomplete
            CODUNC = 1003;   --- Segment may or may not be complete
    lntp[3] - Visibility; one of the values
            CODVIS = 1006;   --- Line segment is visible
            CODINV = 1007;   --- Line segment is invisible
            CODUNV = 1008;   --- Visibility of line segment is unknown
            CODDRV = 1009;   --- Line segment drafted visible
            CODISH = 1022;   --- Line segment is invisible
                                 (hidden by own body occ)
    lntp[4] - Smoothness; one of the values
            CODSMO = 1014;   --- Edge is "smooth"
            CODNSM = 1015;   --- Edge is not "smooth"
            CODUNS = 1016;   --- Edge "smoothness" is unknown
            CODSMS = 1017;   --- Edge "smooth" but coincident
                                 with silhouette
    lntp[5] - Internal edge; one of the values
            CODINE = 1018;   --- Edge is internal
            CODNIN = 1019;   --- Edge is not internal
            CODINU = 1020;   --- Not known whether edge is internal
            CODINS = 1021;   --- Edge is internal, coincides with
                                 silhouette
            CODIGN = 1023;   --- Edge lies on boundary of ignorable
                                 feature

    If segtyp is SGTPGC, SGTPGS or SGTPGB the lntp[2..5] are always

            lntp[2] = CODUNC = 1003; --- Segment may or may not be
                                         complete
            lntp[3] = CODUNV = 1008; --- Visibility of segment is unknown
            lntp[4] = CODUNS = 1016; --- "smoothness" is unknown
            lntp[5] = CODINU = 1020; --- Not known whether internal


    If regional data was requested in a call to PK_TOPOL_render_line
```

```
        lntp[6] and lntp[7] contain start and end point indices for
        SGTPED and SGTPSI segments. The indices uniquely identify the
        image points at each end of the segment.

    If the geometry type is L3TPF* :

        lntp[2]               - Number of loops in facet.

        lntp[3], lntp[4], ... - The number of vertices in each loop.

    If the geometry type is L3TPT* :

        lntp[2] - Number of vertices in the facet strip.

    If the geometry type is L3TPPC or L3TPRC:

        lntp[8] is the degree of the parametric curve

    If the geometry type is L3TPNC or L3TPRN:

        lntp[8]  is the degree of the NURBs curve
        lntp[9]  is the number of b-spline vertices
        lntp[10] is the number of knots

    If segtyp type is SGTPMF there is one loop consisting of three
    vertices.

    If segtyp type is SGTPSI then the last element in the ltp array is
    the silhouette label.


    ifail:

    'ifail' - a code indicating whether the Frustrum wants
              to abort graphic output.

              One of the values:

              CONTIN = 0;      --- Continue: no errors
              ABORT  = -1011;  --- Abort: return control to caller
```

PK_GROUP_array_t

This structure contains an array of entities of type PK_GROUP_t

```
typedef struct PK_GROUP_array_s
    {
    PK_GROUP_t     *array;   --- array of GROUPs
    int             length;  --- length of array
    }
    PK_GROUP_array_t;

It has the fields:

'array':   array of entities of type PK_GROUP_t

'length':  the number of entities in 'array'
```

PK_GROUP_t

This type represents a GROUP. A group is a collection of entities within a part.

A group is a type of entity (PK_CLASS_group is a subclass of PK_CLASS_entity).

PK_HAND_t

```
typedef int PK_HAND_t;
```

PK_HAND_t has just two values:

PK_HAND_left_c	left hand
PK_HAND_right_c	right hand

PK_ICURVE_array_t

This structure contains an array of entities of type PK_ICURVE_t

```
typedef struct PK_ICURVE_array_s
    {
    PK_ICURVE_t    *array;   --- array of Intersection CURVEs
    int             length;  --- length of array
    }
    PK_ICURVE_array_t;
```

It has the fields:

'array': array of entities of type PK_ICURVE_t

'length': the number of entities in 'array'

PK_ICURVE_t

This type represents an Intersection CURVE. An intersection curve is a connected subset of the intersection between two surfaces.

An intersection curve is a type of curve (PK_CLASS_icurve is a subclass of PK_CLASS_curve).

An intersection curve has no standard form. Intersection curves are created when surfaces are intersected and the intersection cannot be represented by one of the analytic curves.

PK_identify_details_r_t

```
struct PK_identify_details_r_s
    {
    int               n_facesets; --- number of facesets
    PK_FACE_array_t   *facesets;   --- facesets
    PK_detail_t       *details;    --- detail type of each faceset
    };
typedef struct PK_identify_details_r_s PK_identify_details_r_t;
```

This structure contains the results of a PK_FACE_identify_details operation.

It contains the fields:

'n_facesets' The number of facesets found

'facesets' The facesets

'details' A token of type PK_detail_t indicating the type of detail that the corresponding faceset comprises.

PK_identify_facesets_r_t

```
struct PK_identify_facesets_r_s
    {
    int                n_survive_target_faces; --- number of surviving
                                               --- facesets on target
    PK_FACE_array_t   *survive_target_faces;   --- surviving facesets
                                               --- on target
    int                n_survive_tool_faces;   --- number of surviving
                                               --- facesets on tool
    PK_FACE_array_t   *survive_tool_faces;     --- surviving facesets
                                               --- on tool
    int                n_reject_target_faces;  --- number of rejected
                                               --- facesets on target
    PK_FACE_array_t   *reject_target_faces;    --- rejected facesets
                                               --- on target
    int                n_reject_tool_faces;    --- number of rejected
                                               --- facesets on tool
    PK_FACE_array_t   *reject_tool_faces;      --- rejected facesets
                                               --- on tool
    PK_EDGE_array_t    bound_target_edges;     --- bounds of facesets
                                               --- on target
    PK_EDGE_array_t    bound_tool_edges;       --- bounds of facesets
                                               --- on tool
    };
typedef struct PK_identify_facesets_r_s PK_identify_facesets_r_t;
```

This structure contains the results of an identify facesets operation. The PK memory associated with this structure may be freed by calling PK_identify_facesets_r_f.

Description of fields:

'n_survive_target_faces' The number of surviving facesets on the target body.

'survive_target_faces' The surviving facesets on the target body. Each array element represents a complete faceset.

'n_survive_tool_faces' The number of surviving facesets on the tool body.

'survive_tool_faces' The surviving facesets on the tool body. Each array element represents a complete faceset.

'n_reject_target_faces' The number of rejected facesets on the target body.

'reject_target_faces' The rejected facesets on the target body. Each array element represents a complete faceset.

'n_reject_tool_faces' The number of rejected facesets on the tool body.

'reject_tool_faces' The rejected facesets on the tool body. Each array element represents a complete faceset.

'bound_target_edges' The boundary edges that divide the surviving and rejected facesets on the target body.

```
'bound_tool_edges'        The boundary edges that divide the surviving
                          and rejected facesets on the tool body.
```

PK_imprint_r_t

```
struct PK_imprint_r_s
    {
    PK_EDGE_array_t    target_edges;     --- corresponding edges on target
    PK_EDGE_array_t    tool_edges;       --- corresponding edges on tool
    PK_VERTEX_array_t  target_vertices;  --- corresponding vertices on target
    PK_VERTEX_array_t  tool_vertices;    --- corresponding vertices on tool
    };
typedef struct PK_imprint_r_s PK_imprint_r_t;
```

This structure contains the results of an imprint operation for bodies or faces.

The arrays of entities are returned under the control of the option structure passed to the imprint function. The arrays will be set to (NULL, 0) if not requested, or empty (length of zero).

The function PK_imprint_r_f may be used to free associated memory.

Description of fields:

'target_edges' The array of corresponding target edges.

'tool_edges' The array of corresponding tool edges.

'target_vertices' The array of corresponding target vertices.

'tool_vertices' The array of corresponding tool vertices.

PK_INSTANCE_array_t

This structure contains an array of entities of type PK_INSTANCE_t

```
typedef struct PK_INSTANCE_array_s
    {
    PK_INSTANCE_t  *array;   --- array of INSTANCEs
    int             length;  --- length of array
    }
    PK_INSTANCE_array_t;
```

It has the fields:

'array': array of entities of type PK_INSTANCE_t

'length': the number of entities in 'array'

PK_INSTANCE_sf_t

```
struct PK_INSTANCE_sf_s
    {
    PK_ASSEMBLY_t   assembly;   --- owning assembly
    PK_TRANSF_t     transf;     --- transform (must be a rigid motion, or
                                ---     PK_ENTITY_null)
    PK_PART_t       part;       --- instanced part
    };

typedef struct PK_INSTANCE_sf_s PK_INSTANCE_sf_t;
```

A PK_INSTANCE_sf_t contains the definition of an INSTANCE. If the transform is PK_ENTITY_null, then it is treated as the identity transform.

PK_INSTANCE_t

This type represents an INSTANCE. An instance positions and orientates a single occurrence of a part within an assembly.

An instance is a type of topology (PK_CLASS_instance is a subclass of PK_CLASS_topol).

PK_intersect_fc_t

typedef int PK_intersect_fc_t;

This datatype contains information that classifies how a face intersects a bounded curve.

One of these values:

PK_intersect_fc_simple_c	The curve passes through the surface of the face at a single point.
PK_intersect_fc_tangent_c	The curve touches the surface of the face at single point but does not pass through it.
PK_intersect_fc_out_in_c	The curve passes from being outside the face to the interior of the face.
PK_intersect_fc_in_out_c	The curve passes from being in the interior of the face to the outside of the face.
PK_intersect_fc_out_coi_c	The curve passes from being outside the face to a region of coincidence with the boundary of the face.
PK_intersect_fc_coi_out_c	The curve passes from a region of coincidence with the boundary of the face to being outside the face.
PK_intersect_fc_coi_in_c	The curve passes from a region of coincidence with the boundary of the face to being inside the interior of the face.
PK_intersect_fc_in_coi_c	The curve passes from being inside the interior of the face to a region of coincidence with the boundary of the face.
PK_intersect_fc_in_tangent_c	The curve is tangent to the inside of an edge or passes through a vertex, but remains outside the face.
PK_intersect_fc_out_tangent_c	The curve is tangent to the outside of an edge or passes through a vertex, but remains outside the face.
PK_intersect_fc_in_c	The start or end of the curve is inside the face or lies in its boundary.
PK_intersect_fc_start_c	The curve enters the face at the start of a region of coincidence with the surface.
PK_intersect_fc_end_c	The curve leaves the face at the end of a region of coincidence with the surface.

PK_intersect_fc_simple_c and PK_intersect_fc_tangent_c are used only when curve and the surface of the face intersect at a single point. All other values are used when all or part of the curve lies in the surface.

PK_intersect_vector_t

typedef int PK_intersect_vector_t;

```
    PK_intersect_vector_simple_c    simple intersection
    PK_intersect_vector_start_c     start of coincidence
    PK_intersect_vector_end_c       end of coincidence
    PK_intersect_vector_tangent_c   tangency
```

PK_INTERVAL_t

```
struct PK_INTERVAL_s
    {
    double value[2];
    };
typedef struct PK_INTERVAL_s PK_INTERVAL_t;
```

This structure represents a real interval.

Specific Errors

```
    PK_ERROR_bad_interval           the interval is not a valid subset of
                                    a curve's parameter space
```

The range represented is that which starts at 'value[0]' and goes in the direction of increasing value to 'value[1]'.
'value[0]' must be less than or equal to 'value[1]'.
When the two values are equal, the interval has no extent.

When the interval is part of a curve's parameter space, 'value[0]' must lie in the curve's parameter range as given by PK_CURVE_ask_interval. If the curve is non-periodic, then 'value[1]' must also lie in this range. If the curve is periodic, 'value[1]' will be in the period above 'value[0]' when the required interval crosses the seam.

PK_knot_type_t

```
typedef int PK_knot_type_t;
```

This field can take any one of the following values.

```
    'PK_knot_unset_c'           : The type has not been calculated.
    'PK_knot_non_uniform_c'     : The knot set is general non uniform.
    'PK_knot_uniform_c'         : The knots are uniformly spaced.
    'PK_knot_quasi_uniform_c'   : The first and last knots have
                                  multiplicity degree+1, and the
                                  remainder have multiplicity 1, with
                                  equal spacing.
    'PK_knot_piecewise_bezier_c' : The first and last knots have
                                  multiplicity degree+1 and the
                                  remainder have multiplicity degree,
                                  with equal spacing.
    'PK_knot_bezier_ends_c'     : The first and last knots have
                                  multiplicity degree+1.  In other
                                  respects the knot set is non uniform.
    'PK_knot_smooth_seam_c'     : Applicable only to PK_BCURVE_create.
                                  Modifications will be made to the curve
                                  input data to guarantee the end and start
                                  of the curve meet sufficiently smoothly to
                                  ensure periodic parameterisation. This is
                                  done by repeating the first degree vertices
                                  at the end and by matching knot intervals.
```

PK_LAW_sf_t

```
struct PK_LAW_sf_s
    {
    int                  degree;          --- The degree = order-1.      (0)
    int                  n_vertices;      --- The number of vertices.    (0)
    int                  vertex_dim;      --- The dimension of each vertex.
                                          --- (1)
    PK_LOGICAL_t         is_rational;     --- If the law is rational.
                                          --- (PK_LOGICAL_false)
    double              *vertex;          --- The vertices.          (NULL)
    int                  n_knots;         --- The number of knot values. (0)
    int                 *knot_mult;       --- The multiplicity of each knot.
                                          --- (NULL)
    double              *knot;            --- The distinct knot values.
                                          --- (NULL)
    PK_knot_type_t       knot_type;       --- Enum describing the knot set.
                                          --- (PK_knot_unset_c)
    PK_LOGICAL_t         is_periodic;     --- If the law is periodic.
                                          --- (PK_LOGICAL_false)
    PK_LOGICAL_t         is_closed;       --- If the law is closed
                                          --- (PK_LOGICAL_false)
    };
typedef struct PK_LAW_sf_s PK_LAW_sf_t;
```

This data structure is the standard form for a law function.

Specific Errors

```
PK_ERROR_bad_value              unreadable value
PK_ERROR_bad_knots              illegal knot multiplicity
                                or knot set not monotonic increasing
PK_ERROR_wrong_number_knots     knots inconsistent with vertices
PK_ERROR_bad_dimension          illegal 'vertex_dim'
PK_ERROR_weight_le_0            one of the weights is zero or negative


'vertex dim':

    This field gives the number of doubles per vertex in the array
    'vertex'.

'is_rational':

    This field should be set to 'PK_LOGICAL_true' if the law is rational.
    In this case the vertex_dim field must be one greater than the dimension
    of the law, and the weights must be included in the 'vertex' array.

'vertex':

    If the law is polynomial, i.e. non-rational, of dimension n, then the
    values in this filed represent the vertices explicitly, in the order
    [f1,1; f2,1; ...; fn,1; f1,2; f2,2; ...].  If it is rational then the
    cartesian points are multiplied by the weights, so that the values
    in this field represent [w1f1,1; w1f2,1; ...; w1fn,1; w1; w2f1,1; w2f2,2;
    ...].

'knot_mult' and 'knot':

    The values in 'knot_mult' give the number of times each knot is to be
    repeated. The minimum multiplicity allowed for any knot is 1.
    The maximum multiplicity allowed other than for the first and last knot is
    'degree'.  The maximum allowed for the first or last knot is 'degree'+1.

    The values in 'knot' must be distinct and form a monotone increasing set.
```

See the documentation below for an explanation of the required total number of knots.

'knot_type':

See the documentation for 'PK_knot_type_t'.

'is_periodic':

If this field is set to 'PK_LOGICAL_true', the parametrisation of the law "wraps around". It implies that the law has equal value and first derivative continuity at the seam.

See the documentation below for additional requirements for the knot set of a periodic law.

'is_closed':

This field being set to 'PK_LOGICAL_true' does not necessarily imply that the parametrisation is periodic. It indicates that the law values at the ends of the paramter range are equal.

The number of knots:

The knot set referred to here is the "expanded knot set" obtained by repeating each value in the array 'knot' the number of times given by the corresponding element in 'knot_mult'.

The law can the considered to be defined in terms of a NURBS curve of degree n defined over k parameters (k-1 parameter intervals). This requires a further n parameters at either end of its knot set in order for the necessary k+n-1 b-spline basis functions to be fully defined. These additional parameters are commonly known as the "imaginary knots". Thus the total number of knots p is related to the total number of basis functions and hence the number m of control vertices by the expression p = m+n+1.

Periodic laws:

For a periodic law the "expanded knot set" must "wrap around" in the sense that the intervals between the first n+1 knots (n "imaginary" knots and one "real" knot) must be the same as the corresponding intervals between the last n+1 "real" knots, and the intervals between the last n+1 knots (one "real" knot and n "imaginary" knots) must be the same as those between the first n+1 "real" knots.

The "period" of such a law is the parameter interval separating the first and last "real" knots.

if indices begin at 0:

$$\text{period} = t_{k+n-1} - t_{n}$$

for i = 0 to n:

$$t_{k+n+i-1} - \text{period} = t_{n+i}$$

$$t_{i} + \text{period} = t_{k+i-1}$$

In addition a periodic law must be so constructed as to preserve G1 continuity (continuity of tangent direction) at the join.

PK_LINE_array_t

This structure contains an array of entities of type PK_LINE_t

```
typedef struct PK_LINE_array_s
    {
    PK_LINE_t      *array;   --- array of LINEs
    int             length;  --- length of array
    }
    PK_LINE_array_t;
```

It has the fields:

'array': array of entities of type PK_LINE_t

'length': the number of entities in 'array'

PK_LINE_sf_t

```
struct PK_LINE_sf_s
    {
    PK_AXIS1_sf_t basis_set; --- a cartesian point and a direction
    };

typedef struct PK_LINE_sf_s PK_LINE_sf_t;
```

This is the standard form for an unbounded straight line which passes through the given 'location' in the given 'direction'.

The parameterisation of the line is as follows:

```
    P=basis_set.location
    V=basis_set.direction

    P( t ) = P + t*V
```

PK_LINE_t

This type represents an unbounded straight LINE.

A line is a type of curve (PK_CLASS_line is a subclass of PK_CLASS_curve).

The line is defined by data contained in a PK_LINE_sf_t

PK_local_change_topol_t

```
typedef int PK_local_change_topol_t;
```

This data type indicates if topology changes are permitted in the local operation. It has two values:

```
    PK_local_change_topol_no_c         don't check any surfaces
    PK_local_change_topol_yes_c        check all surfaces
```

PK_local_check_fa_fa_t

```
typedef int PK_local_check_fa_fa_t;
```

This data type controls the checking for face-face inconsistencies during a lop operation. It has two values:

```
    PK_local_check_fa_fa_no_c     don't do any face-face inconsistency checks
    PK_local_check_fa_fa_yes_c    do all of the face-face inconsistency checks
```

PK_local_check_su_X_t

typedef int PK_local_check_su_X_t;

This data type selects which of the new surfaces will be checked for self-intersections during the local operation. It has two values:

PK_local_check_su_X_no_c	don't check any surfaces
PK_local_check_su_X_yes_c	check all surfaces

PK_local_check_t

typedef int PK_local_check_t;

PK_local_check_no_c	local checking not performed
PK_local_check_ok_c	entity passes local checks
PK_local_check_negated_c	entity has been negated and is now valid
PK_local_check_failed_c	entity fails local checks

PK_local_check_topol_t

typedef int PK_local_check_topol_t;

This data type indicates what type of topology checking is performed on the resultant entity. It has two values:

PK_local_check_topol_no_c	don't check any topology
PK_local_check_topol_yes_c	check all topology

PK_local_status_t

typedef int PK_local_status_t;

This data type indicates the status of the local operation. It has ten values:

PK_local_status_ok_c	operation succeeded
PK_local_status_nocheck_c	operation succeeded, but no checks performed on entity
PK_local_status_fail_c	operation failed
PK_local_status_cant_get_pt_c	failed to locate geometry for vertex
PK_local_status_cant_get_cu_c	failed to locate geometry for edge
PK_local_status_cant_get_su_c	failed to locate geometry for face
PK_local_status_cant_offset_c	failed to offset or face could not be deleted
PK_local_status_side_cu_fail_c	failed to create a side curve
PK_local_status_side_su_fail_c	failed to create a side surface
PK_local_status_not_same_by_c	entities are not from the same body
PK_local_status_fa_fail_c	face check fails
PK_local_status_fa_fa_fail_c	face/face check fails
PK_local_status_ed_remains_c	edge could not be deleted
PK_local_status_point_contact_c	point contact found in sheet body
PK_local_status_bad_reference_c	taper reference entity is not valid
PK_local_status_not_supported_c	given combination of reference entities currently not supported

PK_LOGICAL_t

```
typedef unsigned char PK_LOGICAL_t;
```

This type represents a logical value.

It has the two values:

```
    PK_LOGICAL_true
    PK_LOGICAL_false
```

PK_LOOP_array_t

This structure contains an array of entities of type PK_LOOP_t

```
typedef struct PK_LOOP_array_s
    {
    PK_LOOP_t      *array;   --- array of LOOPs
    int             length;  --- length of array
    }
    PK_LOOP_array_t;
```

It has the fields:

'array': array of entities of type PK_LOOP_t

'length': the number of entities in 'array'

PK_LOOP_close_gaps_o_t

```
struct PK_LOOP_close_gaps_o_s
    {
    int             o_t_version;
    };

typedef struct PK_LOOP_close_gaps_o_s PK_LOOP_close_gaps_o_t;
```

This option structure allows for potential expansion of the close_gaps functionality. It currently has no application-settable fields.

PK_LOOP_t

This type represents a LOOP. A loop is a closed connected set of fins which forms a face boundary.

A loop is a type of topology (PK_CLASS_loop is a subclass of PK_CLASS_topol).

PK_LOOP_type_t

```
typedef int PK_LOOP_type_t;

    PK_LOOP_type_vertex_c       loop is just a vertex without any edges
    PK_LOOP_type_wire_c         loop has no interior, e.g. from a wire body
    PK_LOOP_type_outer_c        simple peripheral loop
    PK_LOOP_type_inner_c        loop is a simple hole
```

```
    PK_LOOP_type_winding_c      winding loop on a periodic surface
                                e.g. a circle on a cylinder or doughnut
    PK_LOOP_type_inner_sing_c   loop is a hole around the surface singularity
                                e.g. chopping the top off a cone
    PK_LOOP_type_likely_outer_c an apparently peripheral loop on a doubly
                                closed surface
    PK_LOOP_type_likely_inner_c an apparent hole in a doubly closed surface
    PK_LOOP_type_unclear_c      a loop dividing a periodic degenerate surface
                                in two ( contains just one pole )
    PK_LOOP_type_error_c        invalid loop or algorithm failure
```

PK_MARK_frustrum_t

```
typedef PK_ERROR_code_t (*PK_MARK_open_f_t)(PK_MARK_t, PK_LOGICAL_t);
typedef PK_ERROR_code_t (*PK_MARK_close_f_t)(PK_MARK_t);
typedef PK_ERROR_code_t (*PK_MARK_write_f_t)(PK_MARK_t, int, const char*);
typedef PK_ERROR_code_t (*PK_MARK_read_f_t)(PK_MARK_t, int, char*);
typedef PK_ERROR_code_t (*PK_MARK_check_f_t)(PK_MARK_t, PK_LOGICAL_t *);
typedef PK_ERROR_code_t (*PK_MARK_delete_f_t)(PK_MARK_t);

struct PK_MARK_frustrum_s
    {
    PK_MARK_open_f_t   open_fn;     --- open a rollmark for read or write
    PK_MARK_close_f_t  close_fn;    --- close a rollmark
    PK_MARK_write_f_t  write_fn;    --- write data to a rollmark
    PK_MARK_read_f_t   read_fn;     --- read data from a rollmark
    PK_MARK_check_f_t  check_fn;    --- check if mark is still valid
    PK_MARK_delete_f_t delete_fn;   --- delete a rollmark
    };
typedef struct PK_MARK_frustrum_s PK_MARK_frustrum_t;
```

This data structure holds pointers to six rollmark functions that the application wishes to register with the PK using PK_MARK_start.

PK_MARK_start_o_t

```
struct PK_MARK_start_o_s
    {
    PK_LOGICAL_t forward; --- whether roll-forward is enabled (PK_LOGICAL_true)
    };
typedef struct PK_MARK_start_o_s PK_MARK_start_o_t;
```

Option structure for PK_MARK_start_o_t

```
The option structure controls whether roll-forward is enabled
```

PK_MARK_t

This type represents a session MARK. A session mark records the state of the session so that the session may subsequently returned to this state by a roll back or forward operation.

PK_mass_bound_t

typedef int PK_mass_bound_t;

How to calculate bounds for mass property calculations

Values can be:

```
PK_mass_bound_no_c           Do not calculate bounds
PK_mass_bound_modulus_c      Bound is a single quantity to add
                             or subtract from the calculated value
PK_mass_bound_interval_c     Bound is an interval specifying the range
                             of the calculated value.
```

The default value is PK_mass_bound_no_c

PK_mass_periphery_t

typedef int PK_mass_periphery_t;

Whether to calculate periphery of the entities.

Values can be:

```
PK_mass_periphery_no_c      Do not calculate periphery of the entities
PK_mass_periphery_yes_c     Calculate periphery of the entities
```

The default value is PK_mass_periphery_yes_c

PK_mass_t

typedef int PK_mass_t;

Controls the level of mass property calculation

Values can be:

```
PK_mass_no_c      Do not find any data in this hierarchy
PK_mass_mass_c    Find the amount and mass of the entities
PK_mass_c_of_g_c  Find the centre of gravity as well
PK_mass_m_of_i_c  Find the moments of inertia as well
```

The default value is PK_mass_m_of_i_c

PK_MEMORY_alloc_f_t

typedef void* (*PK_MEMORY_alloc_f_t)(size_t); --- type for an allocator

This function allocates the specified amount of memory required in bytes.

The function is type compatible with malloc in the standard C run-time library.

PK_MEMORY_free_f_t

typedef void (*PK_MEMORY_free_f_t) (void *); --- type for a free function

This function frees previously allocated memory.

The function is type compatible with free in the standard C run-time library.

PK_MEMORY_frustrum_t

```
struct PK_MEMORY_frustrum_s
    {
    PK_MEMORY_alloc_f_t alloc_fn;                --- registered allocator
    PK_MEMORY_free_f_t free_fn;                  --- registered free function
    };
typedef struct PK_MEMORY_frustrum_s PK_MEMORY_frustrum_t;
```

This data structure holds pointers to the memory handling functions the application wishes to register with the PK using PK_MEMORY_register_callbacks.

The functions should be type compatible with malloc and free in the standard C run-time library.

PK_NABOX_sf_t

```
struct PK_NABOX_sf_s
     {
     PK_AXIS2_sf_t basis_set;    --- coordinate system for box
     PK_BOX_t      box;          --- box
     };
typedef struct PK_NABOX_sf_s PK_NABOX_sf_t;
```

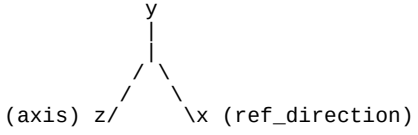
This data structure is the standard form for a non-aligned box

A non-aligned box is a cuboid space defined by an origin, a pair of orthogonal axes (normalised direction vectors) and a set of x, y, z intervals. The term 'non-aligned' means the box axes need not be aligned with the model space axes.

The first axis is called 'the reference direction' (the local X direction).

The second axis is simply called 'the axis'; this is the local Z direction.

The local Y direction is formed from the cross product of the second and first direction vectors i.e. local_Y = local_Z CROSS local_X These form a right handed coordinate system.



The associated box is defined in this local coordinate system. Its extent is defined by x, z and z intervals

PK_neutral_error_t

```
typedef int PK_neutral_error_t;
```

This datatype contains information about the success of a trim operation on a neutral sheet.

Success of the operation is indicated by one of these values:

```
PK_neutral_success_c        No problems were detected and a trimmed
                            neutral sheet has been returned.

PK_neutral_illegal_input_c  Either a neutral sheet was not
                            supplied, or a face was used in
                            more than one pair.

PK_neutral_numerical_failure_c
                            An error occurred while trying to
                            find the trim curves and their
                            endpoints.  The returned neutral sheet
                            was not trimmed correctly.

PK_neutral_unknown_config_c The disposition of pair faces and side
                            faces is not recognised by Parasolid.
                            This is a limitation.

PK_neutral_imprint_failure_c
                            An error occurred while imprinting
                            the trim curves onto the neutral sheet.
                            This often means there is a bug in
                            finding the trim curves.
```

PK_OFFSET_array_t

This structure contains an array of entities of type PK_OFFSET_t

```
typedef struct PK_OFFSET_array_s
    {
    PK_OFFSET_t    *array;   --- array of OFFSET surfaces
    int             length;  --- length of array
    }
    PK_OFFSET_array_t;
```

It has the fields:

'array': array of entities of type PK_OFFSET_t

'length': the number of entities in 'array'

PK_offset_method_t

typedef int PK_offset_method_t;

Sometimes, an offset surface can contain self-intersections. This datatype specifies the method that is used to remove the self-intersections.

If an offset surface contains no self-intersections, the result is a single accurate offset surface regardless of the method you use.

If an offset surface contains self-intersections, it is split along constant parameter lines into regions that contain self-intersections, and regions that contain only accurate surfaces. You can then replace the self-intersecting regions by either:

* Using intersecting neighbouring accurate offset surfaces
* Generating new approximate offset surfaces.

The first of these successfully removes self-intersections, but may leave sharp edges in the final offset surface. The second produces a smoother surface, but parts of the final offset surface will no longer be accurately offset from the original surface.

It has two values :

PK_offset_method_accurate_c	(Default) Self-intersections are replaced by intersecting neighbouring accurate offset surfaces.
PK_offset_method_approximate_c	Self-intersections are replaced by generating approximate offset surfaces that are attached to the neighbouring accurate regions within the given offset tolerance. If an approximate offset surface cannot be generated, the accurate method is used.

PK_OFFSET_sf_t

```
struct PK_OFFSET_sf_s
    {
    PK_SURF_t       underlying_surface;  ---underlying surface
    double          offset_distance;     ---an offset distance
    };

typedef struct PK_OFFSET_sf_s PK_OFFSET_sf_t;
```

Specific Errors

PK_ERROR_invalid_object	an invalid argument has been passed
PK_ERROR_su_self_intersect	the offset surface would self-intersect
PK_ERROR_cant_offset	underlying surface cannot be offset

An offset surface 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'. The 'offset_distance' must be of non-zero length.

The 'offset_distance' is positive in the normal direction as provided by PK_SURF_eval_with_normal and negative in the opposite direction.

For swung, swept and parametric surface types, the surface created will be checked for self-intersections caused by degeneracies on the underlying surface.

PK_OFFSET_t

This type represents an OFFSET surface.

An offset surface is a type of surface (PK_CLASS_offset is a subclass of PK_CLASS_surf).

The offset surface is defined by data contained in a PK_OFFSET_sf_t.

PK_outline_project_t

```
typedef int PK_outline_project_t;
```

PK_outline_project_t can be set to one of the following values:

```
PK_outline_project_no_c     do not project the curves
PK_outline_project_plane_c  project the curves onto a plane perpendicular
                            to the view direction
```

PK_PARAM_bound_t

typedef int PK_PARAM_bound_t;

PK_PARAM_degen_t

typedef int PK_PARAM_degen_t;

This data type is used to specify degeneracies at the beginning or end of a lofted surface.

```
PK_PARAM_degen_no_c                 no degeneracy
PK_PARAM_degen_vector_c             degenerate position vector beyond end
                                    curve supplied
PK_PARAM_degen_bcurve_c             degenerate bcurve beyond end
                                    curve supplied
```

PK_PARAM_direction_t

typedef int PK_PARAM_direction_t;

A direction in parameter space. May be U or V.

```
PK_PARAM_direction_u_c          First parameter direction
PK_PARAM_direction_v_c          Second parameter direction
```

PK_PARAM_end_t

typedef int PK_PARAM_end_t;

This data type is used to specify the end conditions of splinewise bcurves and bsurfs and lofted bsurfs.
It has the following values:

```
PK_PARAM_end_periodic_c             periodic parameterisation
PK_PARAM_end_natural_c              natural end condition ( no curvature )
PK_PARAM_end_clamped_vector_c       end derivative vector(s) supplied
PK_PARAM_end_clamped_bcurve_c       b-spline vertices of derivative
                                    curve supplied
```

PK_PARAM_form_t

typedef int PK_PARAM_form_t;

PK_PARAM_knot_t

typedef int PK_PARAM_knot_t;

This data type is used to specify the knot vector of splinewise
bcurves and bsurfs.
It has the following values:

```
    PK_PARAM_knot_auto_c     knot vector derived from accumulated chord length
    PK_PARAM_knot_defined_c  knot vector is supplied by application
```

PK_PARAM_periodic_t

```
typedef int PK_PARAM_periodic_t;
```

PK_PARAM_sf_t

```
struct PK_PARAM_sf_s
    {
    PK_INTERVAL_t       range;        --- permitted range of parameter values
    PK_PARAM_bound_t    bound[2];     --- token describing entity bounds
    PK_PARAM_periodic_t periodic;     --- token describing entity periodicity
    PK_PARAM_form_t     form;         --- token describing entity form
    PK_LOGICAL_t        continuous;   --- whether continuous
    };
typedef struct PK_PARAM_sf_s PK_PARAM_sf_t;
```

This standard form describes the parameterisation of a geometric entity in one parametric direction.

Description of fields:

'range':
   an interval defining the allowed range of parameter values. If the
   parametric range is infinite, then the values given here will be
   sufficient to include all that part of the geometric entity within the
   size box.

'bound':
   tokens describing the boundedness at either end of the parameter range.
   bound[0] describes the limit range.value[0], and bound[1] describes the
   limit range.value[1]. The token is one of the following:

```
      PK_PARAM_bound_infinite_c    entity extends infinitely
      PK_PARAM_bound_extendable_c  entity may be extended in this direction
      PK_PARAM_bound_bound_c       entity may not be extended
      PK_PARAM_bound_closed_c      ends of parameter range are coincident
      PK_PARAM_bound_degenerate_c  for surfaces only, parameter boundary has
                                   no 3D extent
```

'periodic':
   token indicating whether the parameterisation is periodic. It has one of
   the values:

```
      PK_PARAM_periodic_no_c       not periodic
      PK_PARAM_periodic_yes_c      periodic continuously differentiable at seam
      PK_PARAM_periodic_seamed_c   periodic but not continuously
                                   differentiable across seam
```

'form':
   token indicating the relationship between the parameterisation and 3-space
   extent. It takes one of the following values:

```
      PK_PARAM_form_linear_c       parameter is proportional to distance along
                                   a straight line. For a surface the straight
                                   line corresponds to a constant value of the
                                   other parameter
```

```
      PK_PARAM_form_circular_c     parameter represents an angle round a circle
                                   For a surface, the circle corresponds to a
                                   constant value of the other parameter.
      PK_PARAM_form_any_c          not one of the above

'continuous':
   if PK_LOGICAL_true then all derivatives are continuous, otherwise all
   derivatives are not necessarily continuous.
```

PK_PARAM_twist_t

```
typedef int PK_PARAM_twist_t;

This data type is used to specify whether twist vectors are supplied
It has the following values:

    PK_PARAM_twist_no_c                 twist vectors not supplied
    PK_PARAM_twist_yes_c                twist vectors supplied
```

PK_PART_array_t

This structure contains an array of entities of type PK_PART_t

```
typedef struct PK_PART_array_s
    {
    PK_PART_t      *array;   --- array of PARTs
    int             length;  --- length of array
    }
    PK_PART_array_t;

It has the fields:

'array':   array of entities of type PK_PART_t

'length':  the number of entities in 'array'
```

PK_PART_receive_o_t

```
struct PK_PART_receive_o_s
    {
    int                             o_t_version;
                                        --- version number
    PK_transmit_format_t            transmit_format;
                                        --- (PK_transmit_format_neutral_c)
    PK_LOGICAL_t                    receive_user_fields;
                                        --- (PK_LOGICAL_false)
    };
typedef struct PK_PART_receive_o_s PK_PART_receive_o_t;
```

This option structure is supplied to PK_PART_receive.

```
Description of fields:

'transmit_format'        Format of the 'file'. Permitted values are

                         PK_transmit_format_text_c
                         PK_transmit_format_binary_c
                         PK_transmit_format_neutral_c
                         PK_transmit_format_applio_c
```

'receive_user_fields' PK_LOGICAL_true if user fields are to be received,

PK_PART_t

This type represents a PART. A part is a modelling unit that can be saved and received.

A part is a type of topology (PK_CLASS_part is a subclass of PK_CLASS_topol).

A part is of one of the types:

- PK_ASSEMBLY_t: an assembly
- PK_BODY_t: a body

PK_PART_transmit_o_t

```
struct PK_PART_transmit_o_s
    {
    int                             o_t_version;
                                        --- version number
    PK_transmit_format_t            transmit_format;
                                        --- (PK_transmit_format_neutral_c)
    PK_LOGICAL_t                    transmit_user_fields;
                                        --- (PK_LOGICAL_true)
    int                             transmit_version;
                                        --- (0)
    PK_LOGICAL_t                    transmit_nmnl_geometry;
                                        --- (PK_LOGICAL_false)
    };
typedef struct PK_PART_transmit_o_s PK_PART_transmit_o_t;
```

This option structure is supplied to PK_PART_transmit.

Description of fields:

'transmit_format' Format of the 'file'. Permitted values are

PK_transmit_format_text_c
PK_transmit_format_binary_c
PK_transmit_format_neutral_c
PK_transmit_format_applio_c

'transmit_user_fields' PK_LOGICAL_true if user fields are to be transmitted

'transmit_version' If non-zero, this specifies that the parts are to be transmitted so that they can be received by an earlier version of Parasolid. The integer specifies the version, e.g. 10.1 is 101, 9.0 is 90. The current version is allowed, so the valid values are 120, 111, 110, 101, 100, 91, 90, 80, 71 and 70.

For versions 7.0, 7.1 and 8.0, only one part can be transmitted.

'transmit_nmnl_geometry' If PK_LOGICAL_true, then any nominal geometry on a part is saved, otherwise such geometry is stripped from the transmitted data. This field is ignored if 'transmit_version' is prior to V10.1: nominal geometry is unconditionally removed as earlier versions of Parasolid do not understand nominal geometry.

PK_PARTITION_rcv_deltas_t

```
typedef int PK_PARTITION_rcv_deltas_t;
```

PK_PARTITION_rcv_deltas_t has two values:

```
    PK_PARTITION_rcv_deltas_no_c         do not receive deltas
    PK_PARTITION_rcv_deltas_later_c      retain option to receive deltas later
```

PK_PARTITION_receive_o_t

```
struct PK_PARTITION_receive_o_s
    {
    int                       o_t_version;
                                  --- version number
    PK_transmit_format_t      transmit_format;
                                  --- (PK_transmit_format_neutral_c)
    PK_LOGICAL_t              receive_user_fields;
                                  --- (PK_LOGICAL_false)
    PK_PARTITION_rcv_deltas_t receive_deltas;
                                  --- (PK_PARTITION_rcv_deltas_no_c)
    PK_LOGICAL_t              receive_prev_version_deltas;
                                  --- (PK_LOGICAL_false)
    };
typedef struct PK_PARTITION_receive_o_s PK_PARTITION_receive_o_t;
```

This option structure is supplied to PK_PARTITION_receive.

Description of fields:

'transmit_format' Format of the 'file'. Permitted values are

PK_transmit_format_text
PK_transmit_format_binary
PK_transmit_format_neutral
PK_transmit_format_applio

'receive_user_fields' PK_LOGICAL_true if user fields are to be received

'receive_deltas' Whether to receive no deltas, or to delay receiving deltas until a later call to PK_PARTITION_receive_deltas.

'receive_prev_version_deltas'
Used to indicate that deltas transmitted from earlier compatible versions of Parasolid may be received. This field is ignored if 'receive_deltas' is not set to PK_PARTITION_rcv_deltas_later_c

PK_PARTITION_t

This type represents a PARTITION. A partition is a collection of parts, orphan geometry and transforms which may be rolled back and forward (to recover pre-existing states) independently of other partitions.

PK_PARTITION_transmit_o_t

```
struct PK_PARTITION_transmit_o_s
    {
    int                       o_t_version;
                                  --- version number
    PK_transmit_format_t      transmit_format;
                                  --- (PK_transmit_format_neutral_c)
    PK_LOGICAL_t              transmit_user_fields;
                                  --- (PK_LOGICAL_true)
    PK_PARTITION_xmt_deltas_t transmit_deltas;
                                  --- (PK_PARTITION_xmt_deltas_none_c)
    };
typedef struct PK_PARTITION_transmit_o_s PK_PARTITION_transmit_o_t;
```

This option structure is supplied to PK_PARTITION_transmit.

Description of fields:

'transmit_format'	Format of the 'file'. Permitted values are PK_transmit_format_text PK_transmit_format_binary PK_transmit_format_neutral PK_transmit_format_applio
'transmit_user_fields'	PK_LOGICAL_true if user fields are to be transmitted
'transmit_deltas'	Whether to transmit no deltas, all deltas, or only deltas in the 'main line' from the initial pmark to the current pmark.

PK_PARTITION_xmt_deltas_t

```
typedef int PK_PARTITION_xmt_deltas_t;
```

PK_PARTITION_xmt_deltas_t has three values:

PK_PARTITION_xmt_deltas_none_c	do not transmit deltas
PK_PARTITION_xmt_deltas_all_c	transmit all deltas in partition
PK_PARTITION_xmt_deltas_main_c	transmit all deltas in 'main line' from initial pmark to current pmark

PK_pattern_check_fa_fa_t

```
typedef int PK_pattern_check_fa_fa_t;
```

This data type indicates what type of face/face checking will be done during the patterning operation. It has two values:

PK_pattern_check_fa_fa_no_c	face/face checking not performed
PK_pattern_check_fa_fa_yes_c	face/face checking will be performed

PK_pattern_check_loops_t

```
typedef int PK_pattern_check_loops_t;
```

This data type indicates what type of loop checking will be done during the patterning operation. It has two values:

```
    PK_pattern_check_loops_no_c     loop checking not performed
    PK_pattern_check_loops_yes_c    loop checking will be performed
```

PK_pattern_result_t

typedef int PK_pattern_result_t;

This data type indicates the overall status of the patterning operation. It has seven values:

```
    PK_pattern_result_ok_c             no errors detected in any patterns
    PK_pattern_result_partial_c        errors detected in some patterns
    PK_pattern_result_fail_c           errors detected, no patterns created
    PK_pattern_result_mixed_faces_c    pattern topology not from same body
    PK_pattern_result_instanced_c      pattern topology belong to an instance
    PK_pattern_result_invalid_set_c    cannot pattern given set of topology
    PK_pattern_result_miss_geom_c      pattern topology has missing geometry
```

PK_pattern_status_t

typedef int PK_pattern_status_t;

This data type indicates the status for a particular instance of the patterning operation. It has six values:

```
    PK_pattern_status_ok_c           no errors detected when pattern created
    PK_pattern_status_fail_c         errors detected when pattern created
    PK_pattern_status_outside_fa_c   transformed pattern outside base face
    PK_pattern_status_outside_reg_c  transformed pattern outside base region
    PK_pattern_status_tf_failed_c    transformation of pattern failed, possibly
                                     failed to regenerate SP-curve
    PK_pattern_status_face_clash_c   face-face clash detected
```

PK_pick_approximate_t

typedef int PK_pick_approximate_t;

The 'pick_approx' option can be set to one of the following values:

```
PK_pick_approximate_yes_c          Curves are approximated by a sequence
                                   of straight line chords.

PK_pick_approximate_no_c           Accurate curve geometry is used when
                                   checking whether the ray intersects
                                   a face.
```

The default value is 'PK_pick_approximate_yes_c' and it is recommended for interactive picking operations.

PK_piecewise_rep_t

typedef int PK_piecewise_rep_t;

This data type indicates the basis method of a piecewise representation. It has the following values:

```
    PK_piecewise_rep_bezier_c
    PK_piecewise_rep_polynomial_c
```

```
    PK_piecewise_rep_hermite_c
    PK_piecewise_rep_taylor_c
```

PK_PLANE_array_t

This structure contains an array of entities of type PK_PLANE_t

```
typedef struct PK_PLANE_array_s
    {
    PK_PLANE_t     *array;   --- array of PLANEs
    int             length;  --- length of array
    }
    PK_PLANE_array_t;
```

It has the fields:

'array': array of entities of type PK_PLANE_t

'length': the number of entities in 'array'

PK_PLANE_sf_t

```
struct PK_PLANE_sf_s
    {
    PK_AXIS2_sf_t   basis_set;  --- a point, reference direction and a normal
    };

typedef struct PK_PLANE_sf_s PK_PLANE_sf_t;
```

See documentation for PK_AXIS2_sf_t.

The parameterisation of a plane is as follows:

```
    P = basis_set.location
    x = basis_set.ref_direction
    y = basis_set.axis X basis_set.ref_direction  ( cross product )

    S( u, v) = P + ux + vy
```

PK_PLANE_t

This type represents an unbounded PLANE.

A plane is a type of surface (PK_CLASS_plane is a subclass of PK_CLASS_surf).

The plane is defined by data contained in a PK_PLANE_sf_t.

PK_PMARK_t

This type represents a Partition MARK. A partition mark records the state of a partition so that the partition may subsequently returned to this state by a roll back or forward operation.

PK_POINT_array_t

This structure contains an array of entities of type PK_POINT_t

```
typedef struct PK_POINT_array_s
    {
    PK_POINT_t     *array;   --- array of POINTs
    int             length;  --- length of array
    }
    PK_POINT_array_t;
```

It has the fields:

'array': array of entities of type PK_POINT_t

'length': the number of entities in 'array'

PK_POINT_sf_t

```
struct PK_POINT_sf_s
    {
    PK_VECTOR_t position;    --- cartesian position
    };

typedef struct PK_POINT_sf_s PK_POINT_sf_t;
```

This is the standard form of a POINT which is located at the given position.

Specific Errors

PK_ERROR_bad_position position must be inside size box

PK_POINT_t

This type represents a POINT in 3D space.

A point is a type of geometry (PK_CLASS_point is a subclass of PK_CLASS_geom).

The point is defined by data contained in a PK_POINT_sf_t.

PK_POINTER_t

```
typedef void *PK_POINTER_t;
```

This type represents a pointer to application code or data.

PK_range_1_r_array_t

```
struct PK_range_1_r_array_s
    {
    PK_range_1_r_t  *array;
    int              length;
    };
typedef struct PK_range_1_r_array_s PK_range_1_r_array_t;
```

PK_range_1_r_t

```
struct PK_range_1_r_s
    {
    double          distance;   --- separation between entity and position
    PK_range_end_t  end;        --- end of line of separation on entity
    };
typedef struct PK_range_1_r_s PK_range_1_r_t;
```

This structure records the separation between an entity and a position.

PK_range_2_r_array_t

```
struct PK_range_2_r_array_s
    {
    PK_range_2_r_t  *array;
    int              length;
    };
typedef struct PK_range_2_r_array_s PK_range_2_r_array_t;
```

PK_range_2_r_t

```
struct PK_range_2_r_s
    {
    double          distance;   --- separation between entities
    PK_range_end_t  ends[2];    --- ends of line of separation on entities
    };
typedef struct PK_range_2_r_s PK_range_2_r_t;
```

This structure records the separation between two entities.

PK_range_bound_t

```
struct PK_range_bound_s
    {
    PK_LOGICAL_t        have_upper_bound; --- (PK_LOGICAL_false)
    double              upper_bound;      --- upper bound on min separation
                                          --- (0.0)
    PK_LOGICAL_t        have_lower_bound; --- (PK_LOGICAL_false)
    double              lower_bound;      --- lower bound on min separation
                                          --- (0.0)
    };
typedef struct PK_range_bound_s PK_range_bound_t;
```

This structure is used to place upper and/or lower bounds on the separation of objects.

PK_range_end_t

```
struct PK_range_end_s
    {
    PK_ENTITY_t   entity;         --- entity at end of separation
                                  --- (PK_ENTITY_null)
    PK_ENTITY_t   sub_entity;     --- sub entity at end of separation
                                  --- (PK_ENTITY_null)
    PK_VECTOR_t   vector;         --- end position             (0.0, 0.0, 0.0)
    double        parameters[2];  --- parameter(s) at end position   (0.0, 0.0)
    PK_LOGICAL_t  region;         --- whether solution is regional
                                  --- (PK_LOGICAL_false)
    PK_LOGICAL_t  negative;       --- whether other end is inside body
                                  --- (PK_LOGICAL_false)
    };
typedef struct PK_range_end_s PK_range_end_t;
```

This structure records one end of a line of separation.

If the line of separation terminates on a curve, then 'parameters[0]' will record the curve parameter. If the line of separation terminates on a surface then 'parameters[0]' and 'parameters[1]' will record the surface parameters.

When no 'sub_entity' is identified, this field will be set to PK_ENTITY_null.

PK_range_guess_t

```
struct PK_range_guess_s
    {
    PK_range_guess_type_t  type;            --- form of guess
                                            --- (PK_range_guess_no_c)
    double                 parameters[2];   --- parameters       (0.0, 0.0)
    PK_VECTOR_t            vector;          --- end position (0.0, 0.0, 0.0)
    };
typedef struct PK_range_guess_s PK_range_guess_t;
```

This structure is used to provide an estimate of the position of an end of a line of separation.

If the corresponding entity is a curve or an edge, then an estimate of the curve parameter may be supplied in 'parameters[0]'. If the corresponding entity is a surface or face, then an estimate of the uv surface parameters may be supplied in 'parameters[0]' and 'parameters[1]'. If the corresponding entity is a curve, surface, edge or face then an estimate of the end position may be provided in 'vector'. 'vector' is expected to be on or close to the entity.

PK_range_guess_type_t

```
typedef int PK_range_guess_type_t;
```

PK_range_guess_type_t has three values:

PK_range_guess_no_c	no guess provided
PK_range_guess_param_c	parametric guess provided
PK_range_guess_vector_c	position guess provided

PK_range_result_t

```
typedef int PK_range_result_t;
```

```
PK_range_result_t has three values:
    PK_range_result_found_c          separation successfully found
    PK_range_result_lower_c          minimum separation less than provided
                                     lower bound
    PK_range_result_upper_c          no separation discovered less than
                                     provided upper bound

In cases where a range function's option structure allows bounds to be set on
the separation distance, this data type is used to indicate whether an
allowed solution has been found.
```

PK_REGION_array_t

This structure contains an array of entities of type PK_REGION_t

```
typedef struct PK_REGION_array_s
    {
    PK_REGION_t    *array;   --- array of REGIONs
    int             length;  --- length of array
    }
    PK_REGION_array_t;

It has the fields:

'array':   array of entities of type PK_REGION_t

'length':  the number of entities in 'array'
```

PK_REGION_t

This type represents a REGION. A region is a connected subset of 3-dimensional space.

```
A region is a type of topology (PK_CLASS_region is a subclass of
PK_CLASS_topol).
```

PK_render_bcurve_t

typedef int PK_render_bcurve_t;

```
This field can take any one of the following values.
    'PK_render_bcurve_polyline_c' : output bcurves as polylines (default)
    'PK_render_bcurve_bezier_c'   : output bcurves in Bezier form
    'PK_render_bcurve_nurbs_c'    : output bcurves in NURBS form
```

PK_render_boundary_t

typedef int PK_render_boundary_t;

```
This field can take any one of the following values.
    'PK_render_boundary_no_c'     : don't render surface boundaries
    'PK_render_boundary_yes_c'    : render surface boundaries
```

PK_render_drafting_t

typedef int PK_render_drafting_t;

```
This data type is used to determine whether or not a distinction will be
made between lines hidden by lines and lines hidden by faces.  It has
the values:
    'PK_render_drafting_no_c'     : do not distinguish between lines hidden
                                    by other lines and lines hidden by faces
    'PK_render_drafting_yes_c'    : distinguish between lines hidden by
                                    other lines and lines hidden by faces
```

PK_render_edge_t

```
typedef int PK_render_edge_t;

This field can take any one of the following values.
    'PK_render_edge_no_c'         : don't render edges
    'PK_render_edge_yes_c'        : render edges     (default)
```

PK_render_hierarch_t

```
typedef int PK_render_hierarch_t;

This field can take any one of the following values.
    'PK_render_hierarch_no_c'     : don't produce hierarchical output (default)
    'PK_render_hierarch_no_geom_c': hierarchical output - no geometry
    'PK_render_hierarch_yes_c'    : hierarchical output
    'PK_render_hierarch_param_c'  : hierarchical output parameterised
```

PK_render_ignore_t

```
typedef int PK_render_ignore_t;

PK_render_ignore_t has the following values:

    PK_render_ignore_no_c             no features are ignored
    PK_render_ignore_absolute_c       input an absolute value
    PK_render_ignore_ratio_c          input the ratio of feature to model box
    PK_render_ignore_body_ratio_c     input the ratio of feature to body box
```

PK_render_internal_t

```
typedef int PK_render_internal_t;

This field can take any one of the following values.
    'PK_render_internal_no_c'     : do not indicate whether edges are internal
                                    edges (default)
    'PK_render_internal_yes_c'    : indicate whether edges are internal edges
```

PK_render_invisible_t

```
typedef int PK_render_invisible_t;

This data type is used to determine whether invisible lines will be output
or not.  It has the values:
    'PK_render_invisible_no_c'    : do not output invisible lines
    'PK_render_invisible_yes_c'   : output invisible lines
```

PK_render_memory_target_t

typedef int PK_render_memory_target_t;

The 'memory target' option can be set to one of the following values:

PK_render_memory_target_yes_c	Parasolid will attempt to restrict the memory usage of PK_TOPOL_render_line to the amount of memory given by "memory_target_value". The target value is not guaranteed to be met. The amount of memory used by PK_TOPOL_render_line will be reduced at the expense of total processor time. The value given by 'memory_target' refers to the peak amount of extra memory temporarily allocated during PK_TOPOL_render_line.
PK_render_memory_target_no_c	Parasolid will maximise performance without attempting to restrict memory usage.

PK_render_overlap_t

typedef int PK_render_overlap_t;

PK_render_overlap_t has the following values:

PK_render_overlap_no_c	do not allow for overlapping bodies
PK_render_overlap_yes_c	allow for overlapping bodies

PK_render_param_t

typedef int PK_render_param_t;

This field can take any one of the following values.

'PK_render_param_no_c'	: no parametric hatching (default)
'PK_render_param_attrib_c'	: parametric hatch faces which have parametric hatch attribute
'PK_render_param_free_c'	: parametric hatch using spec in PK_TOPOL_render_line_o_t without locking parametric positions to a point
'PK_render_param_yes_c'	: parametric hatch using spec in PK_TOPOL_render_line_o_t

PK_render_planar_t

typedef int PK_render_planar_t;

This field can take any one of the following values.

'PK_render_planar_no_c'	: no planar hatching (default)
'PK_render_planar_attrib_c'	: planar hatch faces which have planar hatch attribute
'PK_render_planar_free_c'	: planar hatch using vector in 'planar_axis' in PK_TOPOL_render_line_o_t without locking plane positions to a point
'PK_render_planar_yes_c'	: planar hatch using 'planar_axis' in PK_TOPOL_render_line_o_t

PK_render_radial_t

typedef int PK_render_radial_t;

This field can take any one of the following values.

```
    'PK_render_radial_no_c'       : no radial hatching (default)
    'PK_render_radial_attrib_c'   : radial hatch faces which have radial hatch
                                    attribute
    'PK_render_radial_free_c'     : radial hatch using spec in
                                    PK_TOPOL_render_line_o_t without locking
                                    radial positions to a point
    'PK_render_radial_yes_c'      : radial hatch using spec in
                                    PK_TOPOL_render_line_o_t
```

PK_render_region_t

typedef int PK_render_region_t;

This field can take any one of the following values.

```
    'PK_render_region_no_c'       : don't output regional data (default)
    'PK_render_region_attrib_c'   : output regional data for silhouettes and
                                    edges bounding faces with the regional
                                    attribute
    'PK_render_region_yes_c'      : output regional data for all edges and
                                    silhouettes
```

PK_render_self_hidden_t

typedef int PK_render_self_hidden_t;

This data type is used to determine whether or not a distinction will be made between lines hidden by faces of their own occurrence and lines hidden by faces of another occurrence. It has the values:

```
    'PK_render_self_hidden_no_c'  : do not distinguish between lines hidden
                                    by their own or another occurrence
    'PK_render_self_hidden_yes_c' : distinguish between lines hidden
                                    by their own or another occurrence
```

PK_render_silhouette_t

typedef int PK_render_silhouette_t;

This field can take any one of the following values.

```
    'PK_render_silhouette_no_c'   : don't render silhouettes (default)
    'PK_render_silhouette_arcs_c' : render silhouettes, drawing near-circular
                                    analytic silhouette lines as circular arcs
    'PK_render_silhouette_yes_c'  : render silhouettes
```

PK_render_smooth_t

typedef int PK_render_smooth_t;

This field can take any one of the following values.

```
    'PK_render_smooth_no_c'       : don't indicate whether edges are smooth
                                    (default)
    'PK_render_smooth_yes_c'      : indicate whether edges are smooth
    'PK_render_smooth_draft_c'    : draft mode for smooth edge indication
```

PK_render_unfix_t

typedef int PK_render_unfix_t;

This field can take any one of the following values.

```
    'PK_render_unfix_no_c'        : no rendering of unfixed blends (default)
    'PK_render_unfix_attrib_c'    : render unfixed blends as defined by blend
                                    attribute
    'PK_render_unfix_yes_c'       : render unfixed blends using spec in
                                    PK_TOPOL_render_line_o_t
```

PK_render_viewport_t

typedef int PK_render_viewport_t;

This field can take any one of the following values.

```
    'PK_render_viewport_no_c'     : no viewports are supplied (default)
    'PK_render_viewport_yes_c'    : entities outside the single supplied
                                    viewport may be omitted
    'PK_render_viewport_array_c'  : entities outside all the supplied
                                    viewports may be omitted
```

PK_render_vis_t

typedef int PK_render_vis_t;

This data type is used to determine whether any visibility evaluation is to be performed. It has the values:

```
    PK_render_vis_no_c            no visibility evaluated (default)
    PK_render_vis_hid_c           don't output hidden lines
    PK_render_vis_inv_c           output hidden lines marked as invisible
    PK_render_vis_inv_draft_c     output hidden lines and distinguish between
                                  those hidden by other lines and those
                                  hidden by faces.
    PK_render_vis_extended_c      visibility is evaluated.  The information
                                  output is controlled by the 'invisible',
                                  'drafting' and 'self_hidden' options of
                                  the PK_TOPOL_render_line_o_t structure.
```

PK_replace_edge_data_t

```
struct PK_replace_edge_data_s
    {
    int               n_edges;       --- number of edges (0)
    const PK_EDGE_t  *edges;         --- edges to have curves replaced (NULL)
    const PK_CURVE_t *curves;        --- replacement curves (NULL)
    const double     *tolerances;    --- desired edge tolerances (NULL)
    };

typedef struct PK_replace_edge_data_s PK_replace_edge_data_t;
```

This structure is part of the PK_FACE_replace_surfs_o_t options structure. It contains the edge curve replacement data.

Description of fields:

```
'n_edges'           Number of edges to modify. Length of the
                    'edges', 'curves' and 'tolerances' arrays.
                    The default value is 0.
```

'edges'	Array of the edges to replace the curves of. The array default value is NULL.
'curves'	Array of the user-provided replacement curves. The direction of each curve is assumed to be appropriate for the specified edges. No attempt will be made to change the direction of the curves. The array default value is NULL.
'tolerances'	Array of the desired edge tolerances. Values of 0.0 indicate accurate edges. The array default value is NULL.

PK_replace_help_points_t

```
struct PK_replace_help_points_s
    {
    int                n_topologies; --- number of topologies (0)
    const PK_TOPOL_t  *topologies;   --- topologies needing help points (NULL)
    const PK_VECTOR_t *positions;    --- help point positions (NULL)
    };

typedef struct PK_replace_help_points_s PK_replace_help_points_t;
```

This structure is part of the PK_FACE_replace_surfs_o_t options structure. It contains the help point positions to be used to calculate the new geometries of the specified topologies. For each topology, its closest geometry solution to its related help point is prefered to any possible other. The topologies must be of the same type, all edges or all vertices.

Description of fields:

'n_topologies'	Number of topologies to help modify. Length of the 'topologies' and 'positions' arrays. The default value is 0.
'topologies'	Array of the topologies to help modify. All the topologies in this array must be of the same type, edges or vertices. The array default value is NULL.
'positions'	Array of the user-provided help positions for the specified topologies. The array default value is NULL.

PK_replace_merge_t

```
typedef int PK_replace_merge_t;
```

Adjacent faces that were previously non-mergeable may become mergeable when the surface of one or both faces is replaced. This datatype specifies the level of merging that should be performed where feasible. Merging is recommended when adjacent faces are to acquire identical surfaces.

No merging will be attempted for edges for which replacement curves have been provided.

It has the values:

PK_replace_merge_no_c	No replace-specific merging will be performed. Merging may still occur when healing is required to ensure integrity of the final part.
PK_replace_merge_in_c	Faces included in the given set of faces which will be merged together if feasible.

	Vertices on the boundary of the merged face will be merged if feasible, additional faces will not be merged to achieve this.
PK_replace_merge_out_c	Faces included in the given set and faces that are not, but share at least one common edge with faces in the given set, will be merged if feasible. Faces not in the given set and sharing only a vertex with faces in the given set will not be merged. Vertices on the boundary of the merged face will be merged if feasible, additional faces will not be merged to achieve this.

PK_replace_vertex_data_t

```
struct PK_replace_vertex_data_s
    {
    int                n_vertices;   --- number of vertices (0)
    const PK_VERTEX_t *vertices;     --- vertices to have positions
                                     ---                  replaced (NULL)
    const PK_VECTOR_t *positions;    --- vertices replacement positions (NULL)
    const double      *tolerances;   --- desired vertex tolerances (NULL)
    };

typedef struct PK_replace_vertex_data_s PK_replace_vertex_data_t;
```

This structure is part of the PK_FACE_replace_surfs_o_t options structure. It contains the vertex position replacement data.

Description of fields:

'n_vertices'	Number of vertices to modify. Length of the 'vertices', 'positions' and 'tolerances' arrays. The default value is 0.
'vertices'	Array of the vertices to modify. The array default value is NULL.
'positions'	Array of the user-provided replacement positions for the vertices. The array default value is NULL.
'tolerances'	Array of the desired vertex tolerances. Values of 0.0 indicate accurate vertices. The array default value is NULL.

PK_reset_prec_t

```
typedef int PK_reset_prec_t;
```

This datatype indicates the success of an attempt to reset the precision of an edge to the session precision.

One of the following values:

PK_reset_prec_ok_c	geometry successfully replaced and precision set to session precision

PK_reset_prec_tangent_c	geometry not replaced and precision unaltered because surfaces are approximately tangential at edge
PK_reset_prec_missing_geom_c	not enough geometry present on edge or adjacent faces to recompute edge geometry
PK_reset_prec_failure_c	recomputation of edge geometry failed.

PK_section_2_r_t

```
struct PK_section_2_r_s
    {
    PK_section_result_t     result;         --- result of sectioning operation
    int                     n_front_bodies; --- number of front bodies
    PK_BODY_t              *front_bodies;   --- bodies in front of section
    int                     n_back_bodies;  --- number of back bodies
    PK_BODY_t              *back_bodies;    --- bodies behind section
    int                     n_front_faces;  --- number of front faces
    PK_FACE_t              *front_faces;    --- faces in front of section
    int                     n_back_faces;   --- number of back faces
    PK_FACE_t              *back_faces;     --- faces behind section
    int                     n_reports;      --- number of reports
    PK_section_report_r_t  *reports;        --- array of report structures
    };
typedef struct PK_section_2_r_s PK_section_2_r_t;
```

This structure contains the results of a section operation for bodies or faces.

The arrays of entities are returned under the control of the 'fence' field of an options structure, and will be set to (NULL, 0) if not requested, or empty (length of zero).

Description of fields:

'result'	This status field indicates the overall state of the sectioning operation. Permitted values are : PK_section_result_success_c PK_section_result_no_clash_c PK_section_result_no_effect_c PK_section_result_failed_c
'n_front_bodies'	The number of bodies in the 'front_bodies' array.
'front_bodies'	The array of resultant bodies in front of the sectioning tool. If the 'result' field is set to 'PK_section_result_failed_c', the array will be set to null.
'n_back_bodies'	The number of bodies in the 'back_bodies' array.
'back_bodies'	The array of resultant bodies behind the sectioning tool. If the 'result' field is set to 'PK_section_result_failed_c', the array will be set to null.
'n_front_faces'	The number of faces in the 'front_faces' array.
'front_faces'	The array of new faces of the resultant bodies in front of the sectioning tool. If the 'result' field is set to 'PK_section_result_failed_c', the array will be set to null.
'n_back_faces'	The number of faces in the 'back_faces' array.

'back_faces' The array of new faces of the resultant bodies behind the sectioning tool. If the 'result' field is set to 'PK_section_result_failed_c', the array will be set to null.

'n_reports' The number of sectioning reports in the 'reports' array.

'reports' This array may contain supplementary information regarding the outcome of the sectioning operation. For example, if the operation failed, the reports may contain topology related to the failure.

Destroying the returned data structure

When the application wishes to free the memory associated with the results structure, the function 'PK_section_2_r_f' must be used.

PK_section_check_fa_t

```
typedef int PK_section_check_fa_t;
```

PK_section_check_fa_t has the following values:

```
    PK_section_check_fa_no_c          do not perform any face checks
    PK_section_check_fa_yes_c         perform individual face checks
```

PK_section_fence_t

```
typedef int PK_section_fence_t;
```

PK_section_fence_t has three values:

```
    PK_section_fence_front_c           return bodies in front
    PK_section_fence_back_c            return bodies behind
    PK_section_fence_both_c            return all bodies
```

PK_section_r_t

```
struct PK_section_r_s
    {
    PK_BODY_array_t front_bodies;  --- Bodies in front of section
    PK_BODY_array_t back_bodies;   --- Bodies behind section
    PK_FACE_array_t front_faces;   --- Faces in front of section
    PK_FACE_array_t back_faces;    --- Faces behind section
    };
typedef struct PK_section_r_s PK_section_r_t;
```

This structure contains the results of a section operation for bodies or faces.

The arrays of entities are returned under the control of the 'fence' field of an options structure, and will be set to (NULL, 0) if not requested, or empty (length of zero).

The function PK_section_r_f may be used to free associated memory.

Description of fields:

'front_bodies' The array of resultant bodies in front of the sectioning tool.

'back_bodies'	The array of resultant bodies behind the sectioning tool.
'front_faces'	The array of new faces of the resultant bodies in front of the sectioning tool.
'back_faces'	The array of new faces of the resultant bodies behind the sectioning tool.

PK_section_report_r_t

```
struct PK_section_report_r_s
    {
    PK_section_report_t        report;
    int                        n_entities;
    PK_ENTITY_t               *entities;
    };
typedef struct PK_section_report_r_s PK_section_report_r_t;
```

This data structure contains information that pertains to the result of the sectioning operation.

Description of fields:

'report'	This report field indicates the nature of this sectioning report, permitted values are : PK_section_report_failure_c PK_section_report_solid_has_void_c PK_section_report_partial_coi_c PK_section_report_t_sheet_c PK_section_report_non_manifold_c PK_section_report_invalid_match_c PK_section_report_imprint_fail_c PK_section_report_intersect_fail_c PK_section_report_invalid_face_c
'n_entities'	The length of the 'entities' array.
'entities'	This array may contain topological or geometric entities related to the section report. For example, if the 'report' field is returned with 'PK_section_report_invalid_match_c', the 'entities' array will contain the matched regions.

PK_section_report_t

```
typedef int PK_section_report_t;
```

PK_section_report_t has the following values:

PK_section_report_failure_c	general failure
PK_section_report_solid_has_void_c	solid has illegal void
PK_section_report_partial_coi_c	failure due to partial coincidence
PK_section_report_t_sheet_c	operation would result in T-sheet
PK_section_report_non_manifold_c	operation would result in non-manifold body
PK_section_report_invalid_match_c	invalid matched region specified
PK_section_report_imprint_fail_c	face imprinting operation failed
PK_section_report_intersect_fail_c	face clashing operation failed
PK_section_report_invalid_face_c	invalid face detected

PK_section_result_t

```
typedef int PK_section_result_t;
```

PK_section_result_t has the following values:

PK_section_result_success_c	section succeeded
PK_section_result_no_clash_c	target does clash with tool
PK_section_result_no_effect_c	target not affected by tool
PK_section_result_failed_c	section failed

PK_self_intersect_t

```
typedef int PK_self_intersect_t;
```

This field can take any one of the following values.

'PK_self_intersect_unset_c'	: The curve/surface has not been tested
'PK_self_intersect_false_c'	: The curve/surface does not self intersect
'PK_self_intersect_true_c'	: The curve/surface intersects itself

PK_SESSION_applio_t

```
typedef struct PK_SESSION_applio_s
    {
    int (*open_rd)   (int keylen, const char* key, int *strid);
    int (*open_wr)   (int keylen, const char* key, int *strid);

    int (*rd_chars)  (int strid, int n, char               *data);
    int (*rd_bytes)  (int strid, int n, unsigned char      *data);
    int (*rd_shorts) (int strid, int n, short              *data);
    int (*rd_ints)   (int strid, int n, int                *data);
    int (*rd_doubles)(int strid, int n, double             *data);

    int (*wr_chars)  (int strid, int n, const char         *data);
    int (*wr_bytes)  (int strid, int n, const unsigned char *data);
    int (*wr_shorts) (int strid, int n, const short        *data);
    int (*wr_ints)   (int strid, int n, const int          *data);
    int (*wr_doubles)(int strid, int n, const double       *data);

    int (*close)     (int strid, int abort);

    int (*open_uc_rd)(const PK_UCHAR_t *key, int *strid);
    int (*open_uc_wr)(const PK_UCHAR_t *key, int *strid);
    }
    PK_SESSION_applio_t;
```

This data structure holds pointers to the functions the application wishes to register with the PK using PK_SESSION_register_applio. These functions are used during input and output of transmit files with 'application i/o' format.

open_rd: opens a 'file' for reading with the given key. The 'strid' returned identifies the file in later read/write operations. Valid function returns are FR_no_errors, FR_not_found, FR_open_fail.

open_uc_rd: opens a 'file' for reading with the given Unicode key. The 'strid' returned identifies the file in later read/write operations. Valid function returns are FR_no_errors, FR_not_found, FR_open_fail.

rd_xxxx: reads one or more items of data from the given 'file'. Valid returns are FR_no_errors, FR_end_of_file, FR_read_error.

```
open_wr: opens a new 'file' for writing with the given key. The 'strid'
         returned identifies the file in later read/write operations. Valid
         function returns are FR_no_errors, FR_already_exists, FR_disc_full,
         FR_open_fail.

open_uc_wr: opens a new 'file' for writing with the given Unicode key.
         The 'strid' returned identifies the file in later read/write
         operations. Valid function returns are FR_no_errors,
         FR_already_exists, FR_disc_full, FR_open_fail.

wr_xxxx: writes one or more items of data to the given 'file'. Valid returns
         are FR_no_errors, FR_end_of_file, FR_write_error.

close:   closes the given 'file'. If 'abort' is 0, just close it. If 'abort'
         is 1, close and delete the file (this may be given in the case of
         an error during transmit). Valid returns are FR_no_errors,
         FR_close_fail.
```

PK_SESSION_frustrum_t

```
typedef struct PK_SESSION_frustrum_s
    {
    --- macro PK_SESSION_frustrum_o_m initialises all of these fields to NULL
    PK_FSTART_f_t fstart;
    PK_FABORT_f_t fabort;
    PK_FSTOP_f_t  fstop;
    PK_FMALLO_f_t fmallo;
    PK_FMFREE_f_t fmfree;
    PK_GOSGMT_f_t gosgmt;
    PK_GOOPSG_f_t goopsg;
    PK_GOCLSG_f_t goclsg;
    PK_GOPIXL_f_t gopixl;
    PK_GOOPPX_f_t gooppx;
    PK_GOCLPX_f_t goclpx;
    PK_FFOPRD_f_t ffoprd;
    PK_FFOPWR_f_t ffopwr;
    PK_FFCLOS_f_t ffclos;
    PK_FFREAD_f_t ffread;
    PK_FFWRIT_f_t ffwrit;
    PK_FFOPRB_f_t ffoprb;
    PK_FFSEEK_f_t ffseek;
    PK_FFTELL_f_t fftell;
    PK_FGCRCU_f_t fgcrcu;
    PK_FGCRSU_f_t fgcrsu;
    PK_FGEVCU_f_t fgevcu;
    PK_FGEVSU_f_t fgevsu;
    PK_FGPRCU_f_t fgprcu;
    PK_FGPRSU_f_t fgprsu;
    PK_UCOPRD_f_t ucoprd;
    PK_UCOPWR_f_t ucopwr;
    }
    PK_SESSION_frustrum_t;
```

This data structure holds pointers to the functions the application wishes to register with the PK using PK_SESSION_register_frustrum.

The structure can be initialised with the PK_SESSION_frustrum_o_m macro.

PK_SESSION_kernel_version_t

```
typedef struct
    {
    int major_revision;     --- 7
    int minor_revision;     --- 1
    int build_number;       --- 110
    int year;               --- 1996
    int month;              --- 3
    int day;                --- 5
    int hour;               --- 22
    int minute;             --- 1
    int second;             --- 40
    } PK_SESSION_kernel_version_t;
```

The numbers in the comments describe Parasolid V7.1, build 110, compilation starting on 5th March 1996 at 22:01:40 in the evening.

PK_SESSION_receive_o_t

```
struct PK_SESSION_receive_o_s
    {
    int                             o_t_version;
                                        --- version number
    PK_transmit_format_t            transmit_format;
                                        --- (PK_transmit_format_neutral_c)
    PK_LOGICAL_t                    receive_user_fields;
                                        --- (PK_LOGICAL_false)
    };
typedef struct PK_SESSION_receive_o_s PK_SESSION_receive_o_t;
```

This option structure is supplied to PK_SESSION_receive.

Description of fields:

'transmit_format' Format of the 'file'. Permitted values are

PK_transmit_format_text
PK_transmit_format_binary
PK_transmit_format_neutral
PK_transmit_format_applio

'receive_user_fields' PK_LOGICAL_true if user fields are to be received

PK_SESSION_smp_o_t

```
struct PK_SESSION_smp_o_s
    {
    int                 o_t_version;         --- version number
    PK_thread_t         thread_format;       --- (PK_thread_per_processor_c)
    int                 n_threads;           --- (1)
    PK_LOGICAL_t        on_single_processor; --- (PK_LOGICAL_false)
    };
typedef struct PK_SESSION_smp_o_s PK_SESSION_smp_o_t;
```

This option structure is supplied to PK_SESSION_set_smp.

Description of fields:

'thread_format'	In conjunction with the 'n_threads' field, determines the number of threads to be invoked. Permitted values are PK_thread_disabled_c PK_thread_per_processor_c PK_thread_absolute_c
'n_threads'	The number of threads to be invoked for symmetric multi-processing (SMP). The effect of this value is dependent on the value of the 'thread_format' field. This field may only be set to zero if the 'thread_format' field is set to the value, 'PK_thread_disabled_c'.
'on_single_processor'	PK_LOGICAL_true if SMP is to be enabled on single processor platforms.

PK_SESSION_smp_r_t

```
struct PK_SESSION_smp_r_s
    {
    PK_thread_t          thread_format;
    int                  n_threads;
    PK_LOGICAL_t         on_single_processor;
    int                  n_processors;
    };
typedef struct PK_SESSION_smp_r_s PK_SESSION_smp_r_t;
```

This results structure is returned by PK_SESSION_ask_smp.

Description of fields:

'thread_format'	In conjunction with the 'n_threads' field, determines the number of threads that will be invoked. Permitted values are PK_thread_disabled_c PK_thread_per_processor_c PK_thread_absolute_c
'n_threads'	The number of threads to be invoked for symmetric multi-processing (SMP). The effect of this value is dependent on the value of the 'thread_format' field. This field may only be set to zero if the 'thread_format' field is set to the value, 'PK_thread_disabled_c'.
'on_single_processor'	PK_LOGICAL_true if SMP is to be enabled on single processor platforms.
'n_processors'	The number of online processors detected.

PK_SESSION_software_option_t

```
typedef int PK_SESSION_software_option_t;
```

This type is used to denote a particular variant software option.

Permissible values are:

```
    PK_SESSION_SURF_coincide_c
    PK_SESSION_old_isoclines_c
    PK_SESSION_old_blend_bounds_c
```

PK_SESSION_start_o_t

```
struct PK_SESSION_start_o_s
    {
    int     o_t_version;  --- version number of this option structure
    char   *journal_file; --- name of journal file (NULL)
    int     user_field;   --- size of user fields in integers (0)
    int     reserved;     --- for future use (1)
    };
typedef struct PK_SESSION_start_o_s PK_SESSION_start_o_t;
```

Specific Errors

```
    PK_ERROR_bad_filename          journal_file is not a valid file name
    PK_ERROR_bad_user_field_size   user field length is negative or longer
                                   than the maximum
```

This is the option structure for PK_SESSION_start. Its fields are as follows:

```
  journal_file - the name of the required journal file.  If this is set to a
                 non-empty string, then journalling is enabled to a file of
                 the given name.  If it is set to NULL or the empty string
                 then journalling is not enabled.
  user_field   - the size of user fields in integers.  The maximum length is
                 16.  If it is set to zero then no user fields will be
                 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 is the same as that for the
                 session when the part was archived.
```

PK_SESSION_transmit_o_t

```
struct PK_SESSION_transmit_o_s
    {
    int                             o_t_version;
                                        --- version number
    PK_transmit_format_t            transmit_format;
                                        --- (PK_transmit_format_neutral_c)
    PK_LOGICAL_t                    transmit_user_fields;
                                        --- (PK_LOGICAL_true)
    };
typedef struct PK_SESSION_transmit_o_s PK_SESSION_transmit_o_t;
```

This option structure is supplied to PK_SESSION_transmit.

Description of fields:

```
'transmit_format'        Format of the 'file'. Permitted values are

                         PK_transmit_format_text
                         PK_transmit_format_binary
                         PK_transmit_format_neutral
                         PK_transmit_format_applio

'transmit_user_fields'   PK_LOGICAL_true if user fields are to be transmitted
```

PK_SESSION_watch_create_cb_t

typedef void (*PK_SESSION_watch_create_cb_t)(int);

This callback function is registered via the PK_SESSION_watch_tags function. The create callback function is invoked whenever a watched tagged entity is created.

The callback is invoked once the entity has been initialised. No modelling operation should be performed on the tagged entity, but simple enquiries, such as PK_ENTITY_ask_class, will be possible.

PK_SESSION_watch_destroy_cb_t

typedef void (*PK_SESSION_watch_destroy_cb_t)(int);

This callback function is registered via the PK_SESSION_watch_tags function. The destroy callback function is invoked whenever a watched tagged entity is deleted.

The callback is invoked just before the entity is destroyed. No gurantees can be made about the state of the entity and hence no modelling and/or enquiry operation should be performed on or with the entity.

PK_SHELL_array_t

This structure contains an array of entities of type PK_SHELL_t

```
typedef struct PK_SHELL_array_s
    {
    PK_SHELL_t     *array;   --- array of SHELLs
    int             length;  --- length of array
    }
    PK_SHELL_array_t;
```

It has the fields:

'array': array of entities of type PK_SHELL_t

'length': the number of entities in 'array'

PK_SHELL_sign_t

typedef int PK_SHELL_sign_t;

This datatype contains the sign of a shell.

One of these values:

```
   PK_SHELL_sign_positive_c    shell closes off a finite volume
   PK_SHELL_sign_negative_c    shell closes off an infinite volume
   PK_SHELL_sign_open_c        shell does not close off a volume
```

PK_SHELL_t

This type represents a SHELL. A shell is a connected set of oriented faces.

A shell is a type of topology (PK_CLASS_shell is a subclass of PK_CLASS_topol).

PK_SHELL_type_t

```
typedef int PK_SHELL_type_t;
```

This datatype indicates the type of a shell.

One of these values:

```
PK_SHELL_type_acorn_c           shell has a single acorn vertex
PK_SHELL_type_wireframe_c       shell has one or more wireframe edges
PK_SHELL_type_wireframe_free_c  shell has no wireframe edges, but one
                                or more faces
PK_SHELL_type_mixed_c           shell has both wireframe edges and faces
```

PK_SPCURVE_array_t

This structure contains an array of entities of type PK_SPCURVE_t

```
typedef struct PK_SPCURVE_array_s
    {
    PK_SPCURVE_t   *array;   --- array of Surface Parameter CURVEs
    int             length;  --- length of array
    }
    PK_SPCURVE_array_t;
```

It has the fields:

'array': array of entities of type PK_SPCURVE_t

'length': the number of entities in 'array'

PK_SPCURVE_sf_t

```
struct PK_SPCURVE_sf_s
    {
    PK_SURF_t             surf;          --- surface
    PK_CURVE_t            curve;         --- curve
    };

typedef struct PK_SPCURVE_sf_s PK_SPCURVE_sf_t;
```

Specific Errors

```
PK_ERROR_unsuitable_entity      'curve' is not a BCURVE
PK_ERROR_bad_dimension          'curve' is not 2D
PK_ERROR_invalid_geometry       'curve' does not make a single valid
                                spcurve when embedded in 'surf'  (SERIOUS)
```

An spcurve is the 3D curve resulting from embedding a 2D 'curve' in the parameter space of a 'surface'.

The 'curve' must be a 2D BCURVE; that is it must either be a rational bcurve with a vertex dimensionality of 3, or a non-rational bcurve with a vertex dimensionality of 2.

An spcurve may start or end on a surface degeneracy or singularity or it may lie entirely within a degeneracy. An spcurve may not otherwise touch a surface degeneracy.

An spcurve must be G1 continuous and, if periodic, must meet itself with G1 continuity.

PK_SPCURVE_t

This type represents a Surface Parameter CURVE; that is, the curve resulting from embedding a 2D curve in the parameter space of a surface.

An spcurve is a type of curve (PK_CLASS_spcurve is a subclass of PK_CLASS_curve).

The spcurve is defined by data contained in a PK_SPCURVE_sf_t

PK_SPHERE_array_t

This structure contains an array of entities of type PK_SPHERE_t

```
typedef struct PK_SPHERE_array_s
    {
    PK_SPHERE_t    *array;   --- array of SPHEREs
    int             length;  --- length of array
    }
    PK_SPHERE_array_t;
```

It has the fields:

'array': array of entities of type PK_SPHERE_t

'length': the number of entities in 'array'

PK_SPHERE_sf_t

```
struct PK_SPHERE_sf_s
    {
    PK_AXIS2_sf_t basis_set; ---a cartesian point, a ref_direction and a normal
    double        radius;    ---the radius of the sphere(>0)
    };

typedef struct PK_SPHERE_sf_s PK_SPHERE_sf_t;
```

Specific Errors

PK_ERROR_radius_le_0 radius less than or equal to zero

The parameterisation of the sphere is as follows:

```
    C = basis_set.location
    x = basis_set.ref_direction
    y = basis_set.axis X basis_set.ref_direction ( cross product )
    z = basis_set.axis
    R = radius

    S( u, v) = C + R( cos v )( ( cos u )x + ( sin u )y ) + R( sin v )z
```

where (0 <= u < 2PI; -PI/2 <= v < PI/2)

Radius must be larger than zero.

The surface normal points away from the centre.

PK_SPHERE_t

This type represents a SPHERE.

A sphere is a type of surface (PK_CLASS_sphere is a subclass of PK_CLASS_surf).

The sphere is defined by data contained in a PK_SPHERE_sf_t.

PK_SPUN_array_t

This structure contains an array of entities of type PK_SPUN_t

```
typedef struct PK_SPUN_array_s
    {
    PK_SPUN_t      *array;   --- array of SPUN surfaces
    int             length;  --- length of array
    }
    PK_SPUN_array_t;
```

It has the fields:

'array': array of entities of type PK_SPUN_t

'length': the number of entities in 'array'

PK_SPUN_sf_t

```
struct PK_SPUN_sf_s
    {
    PK_CURVE_t            curve;         --- curve to spin
    PK_AXIS1_sf_t         axis;          --- axis defn for spin
    };

typedef struct PK_SPUN_sf_s PK_SPUN_sf_t;
```

Specific Errors

```
    PK_ERROR_unsuitable_entity      curve is not of allowed class
    PK_ERROR_su_self_intersect      surface would be self intersecting
```

This standard form represents a spun surface, that is, the surface obtained by spinning the given 'curve' round the given 'axis'.

The curve must be of one of the following classes:

```
    PK_CLASS_line
    PK_CLASS_circle
    PK_CLASS_ellipse
    PK_CLASS_bcurve
```

The surface is parameterised as:

```
    C = curve
    P = axis.location
    A = axis.axis

    Z(u) = P + (( C(u) - P ).A)A
    R(u) = C(u) - Z(u)

    S(u,v) = Z(u) + R(u)cos(v) + A X R(u)sin(v)
```

The range for u is the parameter range of 'curve', and the range of v is [0, 2Pi]

The curve may not cross the axis, but it may just touch the axis at either end.

PK_SPUN_t

This type represents a SPUN surface.

A spun surface is a type of surface (PK_CLASS_spun is a subclass of PK_CLASS_surf).

The spun surface is defined by data contained in a PK_SPUN_sf_t.

PK_step_surf_t

typedef int PK_step_surf_t;

This data type specifies how step surfaces will be created in step taper.

It has two values

PK_step_tapered_c	Produce tapered step surfaces (default). These surfaces will satisfy the taper condition.
PK_step_normal_c	Produce ruled step surfaces normal to the original relevant taper face. These surfaces will NOT necessarily satisfy the taper condition.

PK_SURF_array_t

This structure contains an array of entities of type PK_SURF_t

```
typedef struct PK_SURF_array_s
    {
    PK_SURF_t      *array;   --- array of SURFaces
    int             length;  --- length of array
    }
    PK_SURF_array_t;
```

It has the fields:

'array': array of entities of type PK_SURF_t

'length': the number of entities in 'array'

PK_SURF_create_blend_o_t

```
struct PK_SURF_create_blend_o_s
    {
    int             o_t_version;    --- version number of option structure
    PK_LOGICAL_t    have_help;      --- (PK_LOGICAL_false)
    PK_VECTOR_t     help;           --- help vector
    PK_LOGICAL_t    have_box;       --- whether box provided (PK_LOGICAL_false)
    PK_BOX_t        box;            --- box of interest
    };

typedef struct PK_SURF_create_blend_o_s PK_SURF_create_blend_o_t;
```

Holds optional data for creating blend surfaces.

The option structure defines data which could help in the creation of a blend surface.

PK_SURF_extend_o_t

```
struct PK_SURF_extend_o_s
    {
    int                 o_t_version;            --- option structure version
    PK_SURF_extension_t extension_type;         --- the type of the extension
                                                --- (PK_SURF_extension_none_c)
    PK_VECTOR_t         extension_point;        --- point to extend to.
    PK_BOX_t            extension_box;          --- box to extend to.
    PK_UVBOX_t          extension_uvbox;        --- uv box to extend to.
    double              u_ratio;                --- ratio of the u parameter
                                                --- range by which to extend
                                                --- the low u boundary (0.0)
    double              U_ratio;                --- ratio of the u parameter
                                                --- range by which to extend
                                                --- the high u boundary (0.0)
    double              v_ratio;                --- ratio of the v parameter
                                                --- range by which to extend
                                                --- the low v boundary (0.0)
    double              V_ratio;                --- ratio of the v parameter
                                                --- range by which to extend
                                                --- the high v boundary (0.0)
    PK_LOGICAL_t        allow_partial_extension;--- if true we will allow
                                                --- partial extensions
                                                --- (PK_LOGICAL_false)
    };

typedef struct PK_SURF_extend_o_s PK_SURF_extend_o_t;
```

Holds optional controls for extending a surface.

The option structure defines options applicable to extending a surface. It contains the following options:

'extension_type' Specifies what type of extension has been requested.

'extension_point' A three-space point to extend the surface to. This field will only be looked at if 'extension_type' has been set to 'PK_SURF_extension_point_c'. The resulting extended surface will not necessarily contain this point.

'extension_box' A three-space box to extend the surface to. This field will only be looked at if 'extension_type' has been set to 'PK_SURF_extension_box_c'.

'extension_uvbox' A parameter box to extend the surface to. This field will only be looked at if 'extension_type' has been set to 'PK_SURF_extension_uvbox_c'.

The next four fields will only be looked at if the 'extension_type' field has been set to 'PK_SURF_extension_ratio_c'. Values must not be less than zero. A value of zero would provide no extension; a value of 0.1 would extend a boundary by one tenth of the existing range of the boundary's parameter.

'u_ratio' An extension of the low u boundary of the surface, expressed as a ratio of the existing u parameter range.

'U_ratio' An extension of the high u boundary of the surface, expressed as a ratio of the existing u parameter range.

'v_ratio' An extension of the low v boundary of the surface, expressed as a ratio of the existing v parameter range.

'V_ratio' An extension of the high v boundary of the surface, expressed as a ratio of the existing v parameter range.

'allow_partial_extension' Whether the surface will still be extended by less than the full extension requested in cases where the full extension would result in an invalid surface.

PK_SURF_extend_status_t

typedef int PK_SURF_extend_status_t;

A status value to be returned when extending a surface. It indicates the result of the surface extension operation.

PK_SURF_extend_ok_c	Extension succeeded
PK_SURF_extend_unextended_c	No extension necessary
PK_SURF_extend_partial_c	Only part of the extension was performed
PK_SURF_extend_invalid_c	Extension failed, surface would be invalid
PK_SURF_extend_failure_c	Internal algorithm failure

PK_SURF_extension_t

typedef int PK_SURF_extension_t;

This data type specifies how a surface is to be extended. It has values:

PK_SURF_extension_none_c	No extension requested
PK_SURF_extension_point_c	Extend to a point.
PK_SURF_extension_box_c	Extend to a box.
PK_SURF_extension_uvbox_c	Extend to a uvbox.
PK_SURF_extension_ratio_c	Extend parameter boundaries by a ratio of the current parameter range.

PK_SURF_intersect_curve_o_t

```
struct PK_SURF_intersect_curve_o_s
    {
    int             o_t_version;    --- version number of option structure
    PK_LOGICAL_t    have_box;       --- whether box provided (PK_LOGICAL_false)
    PK_BOX_t        box;            --- box of interest
    };

typedef struct PK_SURF_intersect_curve_o_s PK_SURF_intersect_curve_o_t;
```

Holds optional controls on intersections between a surface and a curve.

The option structure defines options applicable to the intersection of a surface and a curve.

PK_SURF_intersect_surf_o_t

```
struct PK_SURF_intersect_surf_o_s
    {
    int             o_t_version;    --- version number of option structure
    PK_LOGICAL_t    have_box;       --- whether box provided (PK_LOGICAL_false)
    PK_BOX_t        box;            --- box of interest
    PK_LOGICAL_t    have_uvbox_1;   --- whether box provided for surf_1
                                    --- (PK_LOGICAL_false)
    PK_UVBOX_t      uvbox_1;        --- uvbox for surf_1
    PK_LOGICAL_t    have_uvbox_2;   --- whether box provided for surf_2
                                    --- (PK_LOGICAL_false)
    PK_UVBOX_t      uvbox_2;        --- uvbox for surf_2
    PK_LOGICAL_t    have_vector;    --- whether seed vector provided
                                    --- (PK_LOGICAL_false)
    PK_VECTOR_t     vector;         --- only return branch containing the seed
    };

typedef struct PK_SURF_intersect_surf_o_s PK_SURF_intersect_surf_o_t;
```

Holds optional controls on intersections between two surfaces.

The option structure defines options applicable to the intersection of two surfaces.

PK_SURF_make_cus_isocline_o_t

```
struct PK_SURF_make_cus_isocline_o_s
    {
    int            o_t_version;   --- version number of option structure
    PK_LOGICAL_t   old_iso;       --- whether former isocline curves are
                                  --- to be produced (PK_LOGICAL_false)
    PK_LOGICAL_t   all_points;    --- whether curves using all the control
                                  --- points are to be made (PK_LOGICAL_true)
    };
typedef struct PK_SURF_make_cus_isocline_o_s PK_SURF_make_cus_isocline_o_t;
```

Contains optional controls on isocline curve generation.

Description of fields:

'old_iso' If this flag is set to PK_LOGICAL_true, the former isocline curve behaviour is selected and the returned curves are identical to those

produced using PK_SURF_make_curves_isocline.
When this flag is set to PK_LOGICAL_false, the default value, the more accurate isocline curves are generated.
In the case of analytic isocline curves, this flag as no effect.

'all_points' — In the general case, if this flag is set to PK_LOGICAL_true, the default value, all the control points which can be produced by the isocline calculation are used to generate the curves (NURBS vertices). Otherwise, if this flag is set to PK_LOGICAL_false, only half of them are used.
These latter curves are obviously less accurate than those using all the control points, but use less memory storage space.
In the case of analytic isocline curves, this flag is irrelevant.

PK_SURF_make_sheet_trimmed_o_t

```
struct PK_SURF_make_sheet_trimmed_o_s
    {
    int                   o_t_version;     --- option structure version
    PK_LOGICAL_t          check_wires;     --- identify wire topology
                                           --- (PK_LOGICAL_false)
    PK_LOGICAL_t          check_self_int;  --- check for self-intersection
                                           --- (PK_LOGICAL_false)
    PK_LOGICAL_t          check_loops;     --- check for loop consistency
                                           --- (PK_LOGICAL_false)
    };

typedef struct PK_SURF_make_sheet_trimmed_o_s
               PK_SURF_make_sheet_trimmed_o_t;
```

Holds optional controls for making a sheet body from a surface and a set of sp-curves

The option structure defines options applicable to making a sheet body from a surface and a set of sp-curves

PK_SURF_t

This type represents a SURFace.

A surface is a type of geometry (PK_CLASS_surf is a subclass of PK_CLASS_geom).

A surface is of one of the types:

- PK_BSURF_t: a B geometry surface
- PK_PLANE_t: an infinite plane
- PK_CYL_t: an infinite cylinder
- PK_CONE_t: a semi-infinite cone
- PK_SPHERE_t: a sphere
- PK_TORUS_t: a torus
- PK_OFFSET_t: an offset surface
- PK_SWEPT_t: a swept surface
- PK_SPUN_t: a spun surface
- PK_BLENDSF_t: a blend surface
- PK_FSURF_t: a foreign surface

PK_SURF_trim_data_t

```
struct PK_SURF_trim_data_s
    {
    int                n_spcurves;  --- number of spcurves
    PK_SPCURVE_t      *spcurves;    --- spcurves
    PK_INTERVAL_t     *intervals;   --- spcurve intervals
    int               *trim_loop;   --- which trim loop
    int               *trim_set;    --- which trim set
    };
typedef struct PK_SURF_trim_data_s PK_SURF_trim_data_t;
```

This structure records surface trimming data

Trim data for a surface consists of zero or more trim loops making up zero or more trim sets. Each trim set represents a connected piece of trimmed surface. Each trim loop is made up of trimmed SP-curves. Each trimmed SP-curve is represented by an SP-curve and a parametric interval.

'spcurves' consists of an array of SP-curves which describe the boundary loops of the trimmed surface. More information on 'spcurves' is provided in the documentation of PK_FACE_output_surf_trimmed and PK_SURF_make_sheet_trimmed.

'intervals' is an array of each SP-curve's bounding interval.

Trim sets and trim loops are identified by integers. When a PK_SURF_trim_data_t is returned from PK_FACE_output_surf_trimmed, then trim sets and trim loops will each be allocated identifiers commencing from 0 and increasing by one. When a PK_SURF_trim_data_t is input to PK_SURF_make_sheet_trimmed, there are no limitations on trim loop identifiers.

As an example, if there are two trim_sets and the first has 2 trim_loops, then the fields may be as follows:

spcurves	trim_loop	trim_set
1	0	0
2	0	0
3	0	0
4	0	0
5	1	0
6	1	0
7	1	0
8	2	1
9	2	1
10	2	1
11	2	1

trim_set 0 has trim_loops 0 and 1. trim_loop 0 has spcurves 1 to 4 and trim_loop 1 has spcurves 5 to 7.
trim_set 1 has trim_loop 2 which has spcurves 8 to 11.

PK_SWEPT_array_t

This structure contains an array of entities of type PK_SWEPT_t

```
typedef struct PK_SWEPT_array_s
    {
    PK_SWEPT_t     *array;   --- array of SWEPT surfaces
    int             length;  --- length of array
    }
    PK_SWEPT_array_t;
```

It has the fields:

'array': array of entities of type PK_SWEPT_t

'length': the number of entities in 'array'

PK_SWEPT_sf_t

```
struct PK_SWEPT_sf_s
    {
    PK_CURVE_t            curve;         --- curve to sweep
    PK_VECTOR1_t          direction;     --- vector direction to sweep
    };

typedef struct PK_SWEPT_sf_s PK_SWEPT_sf_t;
```

Specific Errors

PK_ERROR_unsuitable_entity curve is not of allowed class

This standard form represents a swept surface, that is, the surface obtained by sweeping the given 'curve' in the given 'direction'.

The curve must be of one of the following classes:

PK_CLASS_line
PK_CLASS_circle
PK_CLASS_ellipse
PK_CLASS_bcurve

The surface is parameterised as:

C = curve
D = direction

S(u,v) = C(u) + vD

The range for u is the parameter range of 'curve', and the range of v is infinite in each direction.

PK_SWEPT_t

This type represents a SWEPT surface.

A swept surface is a type of surface (PK_CLASS_swept is a subclass of PK_CLASS_surf).

The swept surface is defined by data contained in a PK_SWEPT_sf_t.

PK_taper_method_t

```
typedef int PK_taper_method_t;
```

This data type specifies the method which will be used to construct the taper surfaces of the main taper faces.

It has four values :
PK_taper_method_isocline_c Produces ruled isocline surfaces satisfying the taper condition. If the angle between the taper

	direction and the curve tangent happens to be smaller than the taper angle anywhere along the curve, this method will fail to construct the taper surface. This method produces taper surfaces for mould design.
PK_taper_method_curve_c	Produces ruled surfaces whose ruling direction is in the plane containing the taper direction and the direction perpendicular to the taper direction and the reference edge's curve tangent.
PK_taper_method_surface_c	Produces ruled surfaces whose ruling direction is in the plane containing the taper direction and the tapered face' surface normal.
PK_taper_method_offset_c	Produces ruled surfaces whose ruling direction is that from a point on the reference edge's curve to a point on the "offset curve" on the given "top surface". Viewing from the taper direction, the "offset curve" is a curve that offsets the given distance from the reference edge's curve.

PK_thicken_method_t

typedef int PK_thicken_method_t;

This data type specifies the method which will be used when thickening a sheet body.

It has two values :

PK_thicken_method_offset_c	Thickening a sheet towards its surfaces normal direction
PK_thicken_method_punch_c	Thickening a sheet towards a punch direction given by the user.

PK_thread_t

typedef int PK_thread_t;

PK_thread_t has three values:

PK_thread_disabled_c	SMP disabled
PK_thread_per_processor_c	Number of threads set on a per processor basis
PK_thread_absolute_c	Number of threads set as an absolute value

PK_TOPOL_array_t

This structure contains an array of entities of type PK_TOPOL_t

```
typedef struct PK_TOPOL_array_s
    {
    PK_TOPOL_t     *array;   --- array of TOPOLogical entities
    int             length;  --- length of array
    }
    PK_TOPOL_array_t;
```

It has the fields:

'array': array of entities of type PK_TOPOL_t

'length': the number of entities in 'array'

PK_TOPOL_clash_o_t

```
typedef struct
    {
    int          o_t_version;   --- Version of this PK option structure
    int          n_op_ex;       --- Number of entity pairs that may be ignored
    PK_TOPOL_t  *op_ex1;        --- First half of pairs that may be ignored
    PK_TOPOL_t  *op_ex2;        --- Second half of pairs that may be ignored
                                --- Defaults: 0, NULL, NULL (no ignores)
    PK_LOGICAL_t find_all;      --- Find all clashes, not just the first
    PK_LOGICAL_t find_intersect;--- Examine clashes for true interference
                                --- Defaults: False, False (simplest check)
    PK_LOGICAL_t mul_target_tf; --- There is one transform in tf1 for each
                                --- target entity. Otherwise, the first
                                --- transform is used for all targets.
    PK_LOGICAL_t mul_tool_tf;   --- One transform in tf2 for each tool, as
                                --- per target transforms (above).
                                --- Defaults: False, False (i.e., only one
                                --- transform in each list).
    }
    PK_TOPOL_clash_o_t;
```

Controls for PK_TOPOL_clash: optional excludes, 'find_all' and 'find_intersect' controls, single/ multiple transforms for tools and targets.

'n_op_ex', 'op_ex1' and 'op_ex2' describe a list of entity pairs. 'n_op_ex' is the number of pairs, 'op_ex1' and 'op_ex2' hold lists of first and second elements of pairs respectively.

For each pair (eg, op_ex1[i]' and op_ex2[i]), PK_TOPOL_clash has the option of using the pair to minimise its task; it is at liberty to ignore any clashes that it finds between these two entities. If it ignores these clashes, it won't report them, otherwise it will.

If 'find_all' has the value PK_LOGICAL_false, then the function will return only a single clash (or none, if none exist). It returns the first one that it finds. If 'find_all' has the value PK_LOGICAL_true, it will find and return all the valid clashes that exist.

If 'find_intersect' has the value PK_LOGICAL_true, then for each clash returned, 'clash_type' gives its classification. This allows interference to be distinguished from abutment.

If 'mul_target_tf' has the value PK_LOGICAL_true, then PK_TOPOL_clash assumes that one transform entity has been provided in 'tf1' for each entity in 'targets'. Otherwise, 'tf1' holds only a single transform, which will be applied to all the entities in 'targets'. 'mul_tool_tf' has exactly the same effect with respect to 'tools' and 'tf2'.

PK_TOPOL_clash_t

```
typedef struct
    {
    PK_TOPOL_t              target;         --- Face or Body (aka 'A')
    int                     target_index;   --- Index of owner in input
    PK_TOPOL_t              tool;           --- Face or Body ('B')
```

```
    int                     tool_index;     --- Index of owner in input
    PK_TOPOL_clash_type_t   clash_type;     --- Type of clash being reported
    }
    PK_TOPOL_clash_t;
```

Describes a clash: clashing entities and type of clash

```
Target is an entity from the 'targets' input, Tool is an entity from the
'tools' input, clash_type is a PK_TOPOL_clash_type_t describing the clash.
```

PK_TOPOL_clash_type_t

```
typedef int PK_TOPOL_clash_type_t;
```

Describes the type of clash found.

```
Possible values are:
PK_TOPOL_clash_none                 --- No clash exists
PK_TOPOL_clash_a_in_b               --- Target entity is inside tool entity
PK_TOPOL_clash_b_in_a               --- Tool entity is inside target entity
PK_TOPOL_clash_exists               --- Clash exists; only this is requested
PK_TOPOL_clash_abut_no_class        --- Bounding topologies abut,
                                    --- in/outside unknown
PK_TOPOL_clash_abut_b_in_a          --- Bounding tool topology abuts bounding
                                    --- target topology on the inside
PK_TOPOL_clash_abut_b_out_a         --- Bounding tool topology abuts bounding
                                    --- target topology on the outside
PK_TOPOL_clash_interfere            --- Bounding topologies interfere
```

PK_TOPOL_eval_mass_props_o_t

```
struct PK_TOPOL_eval_mass_props_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_mass_t           mass;           --- required mass properties
                                        --- (PK_mass_m_of_i_c)
    PK_mass_periphery_t periphery;      --- periphery required
                                        --- (PK_mass_periphery_yes_c)
    PK_mass_bound_t     bound;          --- bounds on results required
                                        --- (PK_mass_bound_no_c)
    PK_LOGICAL_t        single;         --- whether to treat topols as
                                        --- single solid (PK_LOGICAL_false)
    PK_LOGICAL_t        use_facets;     --- whether to use alternative facetted
                                        --- method (PK_LOGICAL_false)
    double              facet_tol;      --- chordal tolerance to use for facets
                                        --- 0.0 requests the internal default
    };

typedef struct PK_TOPOL_eval_mass_props_o_s PK_TOPOL_eval_mass_props_o_t;
```

Holds optional controls for evaluating the mass properties of a given set of topological entities.

```
The option structure defines options applicable to evaluating the mass
properties of a given set of topological entities.
```

PK_TOPOL_facet_choice_o_t

```
struct PK_TOPOL_facet_choice_o_s
    {
    int        o_t_version;              --- version of this PK option struct
    PK_LOGICAL_t facet_fin;
    PK_LOGICAL_t fin_fin;
    PK_LOGICAL_t fin_vertex;
    PK_LOGICAL_t vertex_point;
    PK_LOGICAL_t vertex_normal;
    PK_LOGICAL_t vertex_param;
    PK_LOGICAL_t point_vec;
    PK_LOGICAL_t normal_vec;
    PK_LOGICAL_t param_uv;
    PK_LOGICAL_t param_dp;
    PK_LOGICAL_t param_d2p;
    PK_LOGICAL_t facet_face;
    PK_LOGICAL_t facet_occ;
    PK_LOGICAL_t edge_fin;
    PK_LOGICAL_t edge_occ;
    PK_LOGICAL_t error_facet;
    };
    typedef struct PK_TOPOL_facet_choice_o_s PK_TOPOL_facet_choice_o_t;
```

Selection of which tables are returned

Specific Errors

```
    PK_ERROR_o_t_version_incorrect  option structure not initialised correctly
```

PK_TOPOL_choice_o_t
The PK_TOPOL_facet_choice_o_t structure specifies which tables the application wishes PK_TOPOL_facet to return. Each field name of the form

```
    PK_LOGICAL_t <from>_<to>
```

is set by the application to the value PK_LOGICAL_true or PK_LOGICAL_false indicating whether or not the application wishes PK_TOPOL_facet to return this particular table (via the returned PK_TOPOL_facet_r_t structure).

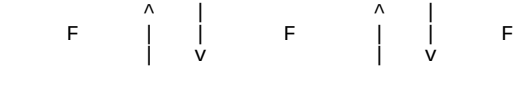
A support macro 'PK_TOPOL_facet_choice_o_m' sets the version field to the current version number of the PK_TOPOL_facet_choice_o_t structure and initialises a sample number of specific fields to PK_LOGICAL_true, sufficient to demonstrate how to return sample data for a simple rendering application.

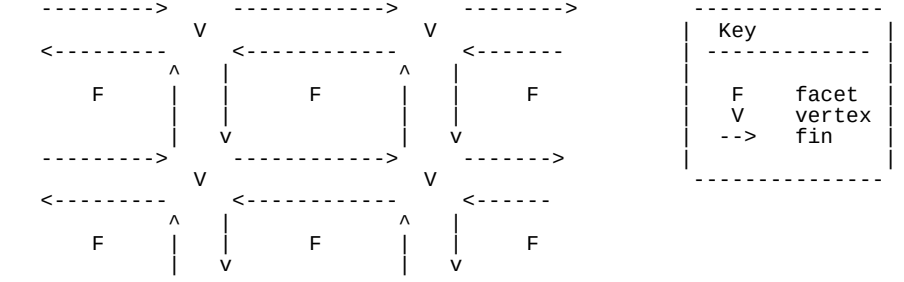
A facet mesh is defined by tables of facet topology, geometry and associative data. Facet topology is described in terms of facet, fin and vertex indices.

The tables of facet vertex data point to tables of coordinate, parameter and derivative data. The associative tables match up facets and fins to faces and edges in the original part.

facets
The outside of a facet is represented by an ordered sequence of 3 or more fins. The fin sequence follows an anti-clockwise order when viewed outside the facet.

example
The anti-clockwise ordering of fins and connected vertex entities is described later as their 'conventional order'. The following diagram illustrates the ordering of fins, viewed from the outside of a quadrilateral facet mesh.

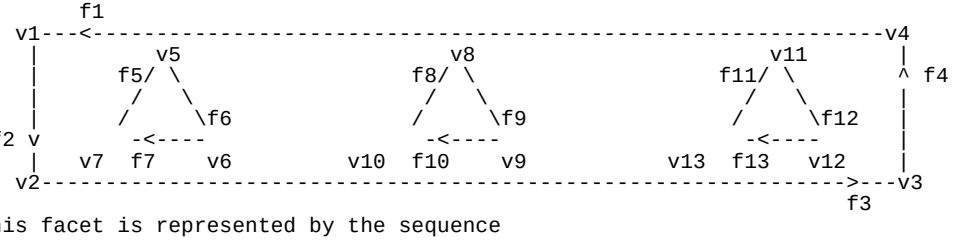




holes in facets
If a facet contains holes (having been created with the 'shape' option set to 'PK_facet_shape_any_c'), it is represented by an ordered sequence of exterior fins, followed by one or more sequences of interior fins, representing holes in the facet. The exterior and interior subsequences of fins are separated from each other by a 'null fin' value of '-1'.

The fin sequence of each interior hole in a face follows an clockwise order when viewed outside the facet. The relative ordering of each interior fin subsequence is arbitrary (other than following the exterior sequence).

example
Example



This facet is represented by the sequence

```
    f1  f2  f3  f4  -1  f5   f6   f7  -1  f8   f9  f10   -1  f11  f12  f13

    \__exterior__/   ^   \_interior_/  ^   \_interior_/   ^    \_interior_/
                     |                 |                 |
                 separator         separator          separator
```

The relative ordering of the interior subsequences "f5 f6 f7", "f8 f9 f10" and "f11 f12 f13" is arbitrary.

Facets will not contain holes (and fin sequences will not contain 'null fin' separators) if the 'shape' option has been set to 'PK_facet_shape_cut_c' or 'PK_facet_shape_convex_c'.

fins
Each facet is assigned an integer value from 0 to (number_of_facets - 1).

A fin is represented by an ordered pair of vertices. Each fin follows an anti-clockwise direction (from its 'tail' to its 'head') consistent with the ordering of fins in a facet. Each fin is assigned an integer value from 0 to (number_of_fins - 1).

Each fin can be associated with a co-fin (one belonging to an adjacent facet). In this case, the co-fin will have a value from 0 to (number_of_fins - 1).

If a fin lies on the perimeter of the facet mesh, it has no co-fin. A null co-fin value of '-1' is used to represent such a case. '-1' is also returned for fins that lie on edges between facetted faces in the case where geometry or trimmed matching have been requested.

A co-fin value of '-2' will be used to represent cases where the associated co-fin for a particular fin has not been found but where a co-fin should exist (i.e. '-1' was not returned in the table). This may be the result of facetting failing on a face (indicated by an entry in the error_facet table), otherwise a co-fin value of '-2' indicates that an error has occurred in fin-fin matching and so the fin-fin table is not correct.

Applications should check explicitly that returned co_fin values lie in this range and handle any exceptions as if the fin has no associated co-fin.

This check will allow for the format of the fin_fin table to be extended in a later release, using fin values greater than or equal to 'number_of_fins' to denote multiply coincident fins. This case will not occur in the current release because general bodies cannot be facetted using fin_fin matching.

It is recommended that the check be added to the application now to avoid the possibility of array bounds errors occurring in the future.

vertices
Each fin is associated with a vertex index, being the vertex at the fin's head. Each vertex is assigned an integer value from 0 to (number_of_vertices - 1).

Facet vertices are of three types; those at original model vertices, those on original model edges and those representing interior points in a face.

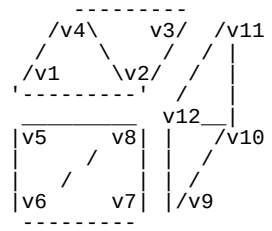
With interior facet vertices, there is one surface normal and a single set of parameter and derivative data at each vertex.

With exterior facet vertices part-way along model edges, there are two surface normals and two sets of parameter and derivative data at each vertex.

With exterior vertices at original model vertices, there are N such normals and N sets of such data for the N faces meeting at the vertex (where N >= 2)

The facet topology tables are defined using separate vertex indices for each model face.

example
The diagram shows the top, front and right hand side faces of a cube with two triangular facets on each face (shown in an 'exploded' schematic form).



facets on the top face are defined as "v1 v2 v4" and "v2 v3 v4"
facets on the front face are defined as "v5 v6 v8" and "v6 v7 v8"
facets on the r.h.s. face are defined as "v9 v10 v12" and "v10 v11 v12")

using the returned subscript
Each vertex index can be used as a subscript into the following tables :

vertex_point	(for the index of its point value)
vertex_normal	(for the index of its surface normal vector)
param_uv, param_dp, param_d2p	(for the index into its parameterisation and derivative data)

shared geometry
Facet vertices from adjacent faces (which have been facetted using the topology matching option) which meet at the same position will refer to the same vertex_point index.

In the example above, the vertex_point entries at vertex indices v2, v8, v12 refer to the same integer value, an index in the 'point_vec' vector table.

This reduces the volume of geometric data output and allows an application to infer this association from vertex indices having the same vertex_point value.

The 'vertex_norm' index is provided because the number of vertex normals is different from the number of vertex point coordinates (there being more than one normal across an model edge or at a model vertex). This level of indirection allows Parasolid to identify cases where vertices of planar facets refer to the same surface normal value and construct the vertex_normal table so that it refers to the same vector entry in the 'normal_vec' table.

The 'vertex_param' index is provided as a common index to the parameter and derivative data tables. Parasolid does not identify common entries in these tables and so these have the same number of entries as there are distinct fin vertices.

mesh topology
Each topology entity (facet, fin or vertex) in the facet mesh is connected to other entities in the mesh. The complete connectivity data are described as the facet mesh topology.

types of tables
The PK_TOPOL_facet function provides a set of "table names" each of which causes a specific set of connectivity or geometric data to be returned from the facet mesh.

one column tables:
This table is implemented as an one dimensional array; with the index value of the <from_facet_entity> as a subscript to an array of <to_face_entity>.

The array length is also returned explicitly, with the tabular data.

two column tables:
This form of table is implemented as an array of structures, with each structure containing two (or more) fields.

If used to represent a 'one to many' mapping (such as facet_fin table) columns are named after the first and second parts of the table name. The tables are returned sorted by numerical order the first column, then by connectivity order using the second column unless specified otherwise.

The other use is returning tables of compound data (such as param_uv).

The array length is returned explicitly, with the tabular data.

PK_TOPOL_facet_mesh_o_t

```
struct PK_TOPOL_facet_mesh_o_s
    {
    int               o_t_version;      --- version of this PK option struct
    PK_facet_shape_t  shape;            --- constraints on facet generation
                                        --- (PK_facet_shape_convex_c)
    PK_facet_match_t  match;            --- vertex matching strategy at edges
                                        --- (PK_facet_match_topol_c)
    PK_facet_density_t density;         --- increased facet silhouette density
                                        --- (PK_facet_density_no_view_c)
    PK_facet_cull_t   cull;             --- culling of view dependent facets
                                        --- (PK_facet_cull_none_c)
    int               n_loops;          --- array of loops in sheet bodies to
    PK_LOOP_t        *loops;            --- be ignored when facetting(0,NULL)
    int               max_facet_sides;  --- max number of sides on a facet (3)
    PK_LOGICAL_t   is_min_facet_width;  --- whether min_facet_width is set
                                        --- (PK_LOGICAL_false)
    double            min_facet_width;  --- minimum facet width
```

```
PK_LOGICAL_t    is_max_facet_width;  --- whether max_facet_width is set
                                     --- (PK_LOGICAL_false)
double             max_facet_width;  --- maximum facet width
PK_LOGICAL_t    is_curve_chord_tol;  --- whether curve_chord_tol is set
                                     --- (PK_LOGICAL_false)
double             curve_chord_tol;  --- chordal tolerance between a curve
                                     --- and its facet edges in model units
                                     --- i.e. upper bound on distance from
                                     --- chord to the curve it approximates
PK_LOGICAL_t    is_curve_chord_max;  --- whether curve_chord_max is set
                                     --- (PK_LOGICAL_false)
double             curve_chord_max;  --- max length of chord in model units
                                     --- i.e upper bound on length of chord
                                     ---  used to approximate a curved edge
PK_LOGICAL_t    is_curve_chord_ang;  --- whether curve_chord_ang is set
                                     --- (PK_LOGICAL_false)
double             curve_chord_ang;  --- max angle between curve and chord
                                     --- used to approximate it, in radians
                                     --- i.e. upper bound on sum of angles
                                     ---  between chord and curve tangents
                                     ---  measured at the chord ends.
PK_LOGICAL_t    is_surface_plane_tol;--- whether surface_plane_tol is set
                                     --- (PK_LOGICAL_false)
double             surface_plane_tol;--- distance tolerance between surface
                                     ---      and its facet, in model units
                                     --- i.e upper bound on distance from a
                                     --- position on a facet to the surface
PK_LOGICAL_t    is_surface_plane_ang;--- whether surface_plane_ang is set
                                     --- (PK_LOGICAL_false)
double             surface_plane_ang;--- angular tolerance between surface
                                     ---      and its facet, in radians.
                                     --- i.e upper bound on the angular
                                     --- deviation between surface normals
                                     --- at any 2 positions under the facet
PK_LOGICAL_t    is_facet_plane_tol;  --- whether facet_plane_tol is set
                                     --- (PK_LOGICAL_false)
double             facet_plane_tol;  --- max dist from facet to mid-plane
                                     --- (defined : passing through C of G)
                                     --- (of facet vertices in a direction)
                                     --- (given by averaged vertex normals)
PK_LOGICAL_t    is_facet_plane_ang;  --- whether facet_plane_ang is set
                                     --- (PK_LOGICAL_false)
double             facet_plane_ang;  --- facet_plane_tol : max_facet_width
                                     --- (max permitted ratio between them)
PK_LOGICAL_t   is_local_density_tol; --- whether local_density_tol is set
                                     --- (PK_LOGICAL_false)
double            local_density_tol; --- local value of surface_plane_tol
                                     --- (for facet silhouette density)
PK_LOGICAL_t   is_local_density_ang; --- whether local_density_ang is set
                                     --- (PK_LOGICAL_false)
double            local_density_ang; --- local value of surface_plane_ang
                                     --- (for facet silhouette density)
PK_facet_degen_t   degen;            --- whether to output multiple
                                     --- vertices at degeneracies and
                                     --- how the singular parameter at the
                                     --- degeneracy is calculated
                                     --- (PK_facet_degen_multiple_vxs_c)
int         n_view_directions;       --- array of view directions in which
PK_VECTOR1_t                         --- to increased the facet density
           *view_directions;         --- (0, NULL)
int         n_local_tols;            --- array of sets of local tolerances
PK_facet_local_tolerances_t
           *local_tols;              --- (0, NULL)
int         n_topols_with_local_tols;--- array of topological entities for
PK_TOPOL_t *topols_with_local_tols;  --- for which local tolerances are
                                     --- specified (0, NULL)
int        *local_tols_for_topols;   --- index into 'local_tols'
```

```
                                        --- picking out the local tolerances
                                        --- to be attached to the
                                        --- corresponding entity in
                                        --- 'topols_with_local_tols'
    PK_facet_ignore_t  ignore;          --- whether to ignore small features
                                        --- when facetting.  Small features
                                        --- can be ignored by inputting an
                                        --- absolute value or a ratio value
                                        --- (PK_facet_ignore_no_c)
    double             ignore_value;    --- features smaller than this will be
                                        --- considered for ignoring (0.0)
    PK_facet_ignore_scope_t
                       ignore_scope;    --- whether to treat a face
                                        --- individually or as part of the
                                        --- face's owning body when looking
                                        --- for ignorable features and
                                        --- facetting a list of faces
                                        --- (PK_facet_ignore_scope_global_c)
    PK_facet_wire_edges_t  wire_edges;  --- whether to take wire edges into
                                        --- account faces when facetting
                                        --- (PK_facet_wire_edges_no_c)
    };
typedef struct PK_TOPOL_facet_mesh_o_s PK_TOPOL_facet_mesh_o_t;
```

Facet mesh generation controls

Specific Errors

PK_ERROR_nitems_lt_0	negative number of items in loop array
PK_ERROR_lt_3_sides	max_facet_sides too small
PK_ERROR_distance_too_small	min/max_facet_width too small
PK_ERROR_distance_too_large	min/max_facet_width too large
PK_ERROR_bad_tolerance	is_<tol> is true but used with a bad <tol> or negative number supplied as for local tolerance
PK_ERROR_unsuitable_topology	entity is not face or body in topols_with_local_tols array
PK_ERROR_bad_value	integer value in local_tols_for_topols array does not lie between 0 and n_local_tols

The PK_TOPOL_facet_mesh_o_t structure provides the following controls :

- facet constraints (i.e. whether holes are permitted in facets or if the generated facets may be concave).
- which edge matching strategy to use when matching adjacent facetted faces
- specifying view dependent options (face culling and silhouette density)
- capping specified loops within a sheet body
- setting tolerance values to be used in the facetted approximation

Setting tolerance values

The tolerance values are defined by pair of PK_LOGICAL_t and double values, the former indicating whether the double value has been set explicitly. PK uses the convention that if no explicit tolerance value is provided, Parasolid is free to calculate its own internal tolerance values. These are generally case specific, rather than using specific fixed values.

Defining a viewing matrix

A number of the mesh options require a viewing matrix to be specified. Parasolid has a number of requirements on such matrices.

In general a viewing matrix shall be of the form M
| R(11) R(12) R(13) Px |

```
      |                           |
M =   | R(21)  R(22)  R(23)  Py   |
      |                           |
      | R(31)  R(32)  R(33)  Pz   |
      |                           |
      | Tx     Ty     Tz     S    |
```

The vector T represents a translation
The vector P represents perspective
S is the inverse global scale vector
The rotation submatrix R defines an orthonormal set of vectors H, V, D

```
      | Hx  Vx  Dx  |
R =   | Hy  Vy  Dy  |
      | Hz  Vz  Dz  |
```

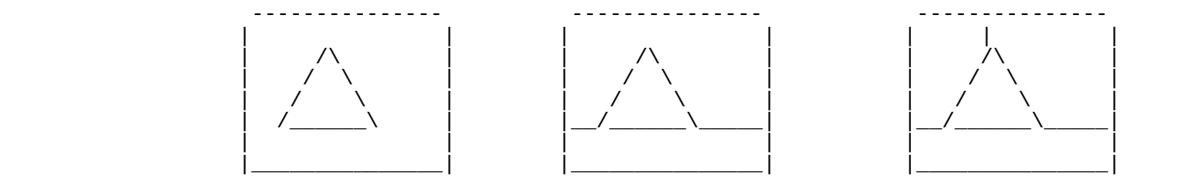
M must satisfy the following constraints:

(i) The rotation matrix R must be orthonormal
(ii) The determinant on R is +1
(iii) D and P must be parallel if P is not a zero vector
(iv) S = 1 - Tz * w (where w = -(D DOT P)

SHAPE

This field controls whether generated facets can contain holes or concave interior angles. The possible settings are one of :

PK_facet_shape_any_c : allow creation of holed or concave facets
PK_facet_shape_cut_c : cut facets with holes into concave facets
PK_facet_shape_convex_c : divide concave facets into convex facets (default)



PK_facet_shape_any_c | PK_facet_shape_cut_c | PK_facet_shape_convex_c

The diagram above illustrates a (contrived) example of a facet containing an interior hole which can be generated using the 'PK_facet_shape_any_c' option.

The constraint 'PK_facet_shape_cut_c' indicates that Parasolid must cut any facets containing interior holes before output. This can sometimes generate cases where the modified facets have concave interior angles.

The constraint 'PK_facet_shape_convex_c' indicates that Parasolid must cut any facets containing holes and further divide any concave facets into convex facets before output.

The PK_facet_shape_cut_c and PK_facet_shape_convex_c constraints are only relevant when max_facet_sides > 3.

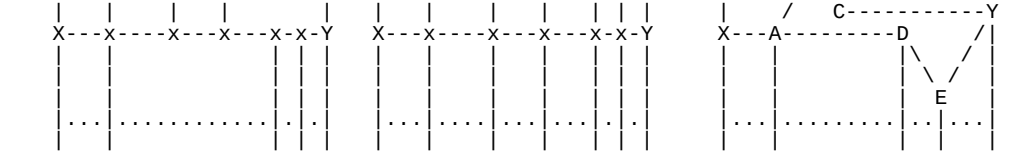
MATCH

This field controls vertex matching between adjacent faces.

The options are

PK_facet_match_geom_c : clip facet boundaries to a common edge
PK_facet_match_topol_c : match facet vertices across a common edge (default)
PK_facet_match_trimmed_c: clip facet boundaries to model edge curve





The above diagram illustrates a (contrived) example of the facets created along an edge XY between two adjacent faces with the various matching options. The dotted lines indicate the curve chord tolerance of the curve on edge XY.

Actual examples of facets created using various matching options can be found in the match section of facetting in the functional description.

The 'PK_facet_match_trimmed_c' option clips exterior facets to within 'curve_chord_tol' of the appropriate model edge but does no matching. If this option is selected, facets will, in general, overlap or leave gaps at the boundaries of adjacent faces. In the above example a gap defined by ABCYEDA exists at the boundary of the two faces.

The 'PK_facet_match_geom_c' option modifies the geometry of such facet meshes by clipping (and possibly extending) exterior facets to meet at the common edge formed by the intersection of the underlying facet planes.

There are certain special cases in which facet meshes may not match these are discussed in the functional description.

If the application inputs a list of faces Parasolid recognises faces that belong to the same body and have identical transform entities, in this case facets will be clipped to common edges unless the option 'ignore_scope' is set to 'PK_facet_ignore_scope_local_c' (see 'ignore_scope' option under IGNORING SMALL FEATURES below).

The 'PK_facet_match_topol_c' option further modifies the facet mesh so that facet vertices are matched across the common edge. The algorithm identifies cases where an exterior vertex on one facet mesh lies between a connected pair of exterior vertices on the other facet mesh and inserts a matching vertex.

DENSITY

The facet density option allows the density of the facet mesh to be increased locally in regions where facet normals change from being front facing to back facing. That is, where facet normals change from being within pi/2 to not being within pi/2 of the view direction specified in 'view_transf' or the view directions given in 'view_directions' if specified.

This option is used when generating view specific facet meshes for use by rendering packages when creating high quality images (containing enhanced detail where the facet surfaces form silhouettes with the view direction).

This field can take the following values :

PK_facet_density_no_view_c : facet density is independent of view (default)
PK_facet_density_use_view_c : increase facet density at view silhouettes

The PK_facet_density_no_view_c refers to no other arguments.

The PK_facet_density_use_view_c argument requires the following values

- the 'view_transf' argument and 'view_directions' argument. If 'view_directions' have been specified these directions will override the single direction specified in the 'view_transf'.

- the 'local_density_tol' tolerance value,

```
  and optionally,

      - the 'local_density_ang' tolerance value

  The 'view_transf' may contain perspective but only the direction component
  will be used.

CULL
  This is a performance enhancement option for use when creating view
  specific images from facet data (possibly in conjunction with the
  the 'PK_facet_density_use_view_c' option).

  It can also be used to reduce the amount of facet data which are output.

  This field can take the following values :

  PK_facet_cull_none_c : do not cull any facets; output them all      (default)
  PK_facet_cull_back_c : cull back facing facets

  The PK_facet_cull_none_c outputs all faces (and refers to no other arguments)

  The PK_facet_cull_back_c argument requires the 'view_transf' to be set to a
  viewing transformation which may contain perspective.

  It is not possible to request (the default) topology matching with this
  option, PK_facet_match_geom_c or PK_facet_match_trimmed_c must be specified.

  This option reduces the size of the generated facet mesh (and consequently
  the amount of topological and geometric data which are output) by not
  facetting back facing faces.  A back facing face is one whose normal at
  every point is backwards facing.  If a face cannot quickly be identified as
  back facing it will be facetted.

  This option has no effect on sheet bodies.

LOOPS
  The 'loops' option allows specific holes in a sheet body to be capped.
  These are specified as an array of interior loops (being ones owned
  by sheet bodies or faces which are included in the 'topol' array').

  These interior loops are ignored during facetting; the effect is
  to extend the surface geometry of the surrounding sheet face across
  the loop.

  The PK_TOPOL_facet_mesh_m macro initialises the loops array to be empty

        n_loops = 0

FACET SIZE AND NUMBER OF SIDES
  The 'max_facet_sides' option controls the maximum number sides that any facet
  can have and must take a value of 3 or more.

  The options 'is_min_facet_width' and 'is_max_facet_width' control the minimum
  and maximum lengths of any of the facets sides.  To set an explicit value
  for such constraints, the associated field
                       min_facet_width
                    or max_facet_width
  is set to a positive value (specified in model units) and the associated flag
  is set to PK_LOGICAL_true.

  In cases where refinement results in a facet being created that has a width
  smaller than 'min_facet_width' Parasolid will disregard the supplied curve,
  surface and planarity tolerances so that the new facet will not be refined
  any further.  Therefore facets may exist with dimensions smaller than
  'min_facet_width'.

CURVE TOLERANCE

  The following arguments control curve tolerance values, measuring how closely
  a curved edge entity is approximated by a number of straight line chords.
```

Three curve tolerance controls are provided :

curve_chord_tol : the maximum chordal distance between a facet edge and its original edge entity

curve_chord_max : the maximum length of any facet edge which represents part of an original edge entity

curve_chord_ang : the maximum angle which is permitted between a facet chord and its original edge entity, measured as the sum of the two angles formed between the chord and the curve tangents at each chord end.

To set one or more of these constraints, the value is set to a positive value and the associated flag set to PK_LOGICAL_true.

SURFACE TOLERANCE

The following arguments control surface tolerance values, measuring how closely any curved surface attached to a face is approximated by a facet mesh.

Two surface tolerance controls are provided :

surface_plane_tol : the maximum distance between the mid-plane of a facet and its original face entity

surface_plane_ang : the maximum angle (in radians) which is permitted between the surface normals at any two positions on the surface which lie within the facet boundary (this is usually met but it is not guaranteed).

PLANARITY TOLERANCE

The "facet to mid-plane" tolerances control how closely vertices lie on a common plane. The 'mid-plane' of a facet used in this calculation is defined to pass through the centre of gravity of the vertex points in the direction of the sum of the vertex normals.

Two planarity tolerance controls are provided :

facet_plane_tol : the maximum distance between the mid-plane of a facet and its facet vertices

facet_plane_ang : the maximum ratio permitted between :

1) the maximum distance from the mid-plane of a facet to any of its facet vertices

2) the maximum width of a facet.

LOCAL FACET DENSITY TOLERANCE

The "silhouette_density" tolerance values are used in conjunction with the 'PK_facet_density_use_view_c' density option. These define local surface tolerance values to be used when generating facets in the region of view dependent facet silhouettes.

Two local facet density tolerance controls are provided:

local_density_tol : the maximum distance between the mid_plane of a facet and its original face entity.

local_density_ang : the maximum angle (in radians) which is permitted between the surface normals at any two positions on the surface which lie within the facet boundary (this is ususally met but it is not guaranteed).

If the PK_facet_density_use_view_c option is requested the 'is_local_density_tol' flag and its associated value must be set (to a positive value) otherwise the option in ignored.

It is also possible (although not compulsory) to set the 'is_local_density_ang' flag, once again the associated value must be positive.

OUTPUT AT DEGENERACIES

The behaviour of the facetting operation at degenerate vertices can be modified via the 'degen' option.

The permitted values for this option:

'PK_facet_degen_multiple_vxs_c': (default)
The function will create multiple vertices at degeneracies. Each facet that is adjacent to a degeneracy will have a unique vertex at the degeneracy. This ensures the correct parameters, normals and derivatives will be returned for the facet at the degeneracy. All of the vertices at the degeneracy will reference the same point.

'PK_facet_degen_single_vx_c':
The function will create single vertices at degeneracies, this vertex can only have one parameter, normal and derivative associated with it, thus all facets adjacent to a degeneracy will have the same value for the parameter, normal and derivative for the vertex at the degeneracy. In this case the facet topology is correct. This option only has an effect on the output returned though the tables in PK_TOPOL_facet, it does not have any effect if used in conjunction with PK_TOPOL_render_facet.

'PK_facet_degen_average_parms_c':
The function will create multiple vertices at degeneracies in the same way as the 'PK_face_degen_multiple_vxs_c' option, in this case when a facet vertex occurs at a degeneracy, the value of the degenerate parameter at the degenerate vertex will be the average of the values of that parameter at the two facet vertices on either side of the degenerate vertex. The normals and derivatives at the degenerate vertex will be evaluated using this average value for the degenerate parameter.

More details and a diagram can be found in vertices at degeneracies

INCREASING SILHOUETTE DENSITY IN MULTIPLE DIRECTIONS

The 'view_directions' option allows the facet density to be increased in multiple view directions. These view directions are specified as an array of unit vectors.

These view directions override the direction given in 'view_transf' as far as increasing the facet density is concerned, that is, the facet density will be increased in all the directions specified in 'view_directions'.

The PK_TOPOL_facet_mesh_m macro initialises the view_directions array to be empty

```
n_view_directions = 0
```

in this case facet density will be increased in the direction specified in the option 'view_transf'.

FACETTING TOPOLOGICAL ENTITIES WITH LOCAL TOLERANCES

It is possible to attach local tolerances to individual or groups of topological entities (either faces or bodies), these local tolerances will then override the facetting tolerances for the particular face/body.

In order to facet certain entities with local tolerances it is necessary to fill in the following fields:

'local_tols'
This field is an array of PK_facet_local_tolerances_t. This array contains all local tolerance sets to be attached to topological entities.

'topols_with_local_tols'
This field takes an array of unique topological entities (faces or bodies) which are to be facetted using local tolerances. If this is empty all

entities will be facetted to the tolerances previously specified in the options structure. If this is not empty the parallel array 'local_tols_for_topols' must be filled in.

'local_tols_for_topols'
This field must be of length 'n_topols_with_local_tols', and takes integer values corresponding to indexes into the 'local_tols' array. For each topological entity in 'topols_with_local_tols', the corresponding entry in 'local_tols_for_topols' is the index into 'local_tols' corresponding to the PK_facet_local_tolerances with which the entity is to be facetted.

When finding the tolerances to use when facetting a paticular face, the tolerances are initially set to the default tolerances. If any tolerances were specified in the options structure these tolerances overwrite the default tolerances. If the face's owning body has local facet tolerances specified these tolerances (provided they are not set to 0.0) override any existing tolerances for this face. If the face itself has local facet tolerances specified these tolerances will override any existing tolerances for this face (provided they are not 0.0).

The curve tolerance along an edge will be the minimum local curve tolerance of the edge's faces.

The PK_TOPOL_facet_mesh_o_m initialises the 'local_tols' array to be empty
n_local_tols = 0,

'topols_with_local_tols' is also initialised to be empty
n_topols_with_local_tols = 0,

in this case all entities will be facetted using the facetting tolerances given above.

IGNORING SMALL FEATURES
The 'ignore' option allows 'small' features to be ignored when facetting. The value specified in 'ignore_value' defines which features are classed as small. The overall box is the box containing all entities to be facetted.

The permitted values for this option:

'PK_facet_ignore_no_c': (default)
All features will be facetted regardless of size.

'PK_facet_ignore_absolute_c':
Any feature with a box size smaller than the value specified in 'ignore_value' may be ignored.

'PK_facet_ignore_ratio_c':
Here 'ignore_value' corresponds to the ratio of box size of the feature to the size of the overall box i.e. if this value is set to 0.01 then features smaller than 1 per-cent of the overall box may be ignored. If facetting bodies the overall box will be the union of all the 'topols' boxes (with 'topol_transfs' applied), if faces are being facetted the overall box will depend on 'local_scope' as described below.

'PK_facet_ignore_body_ratio_c':
'ignore_value' corresponds to the ratio of the box size of the feature to the size of the box of the body containing the feature.

Note that if the option 'loop' has been specified, facetting will ignore the given loops regardless of the value of the 'ignore' option.

The option 'ignore_scope' only has an effect when facetting lists of faces, the permitted values for this option:

'PK_facet_ignore_scope_global_c': (default)
Ensures that the only loops ignored on a face when facetting a list of faces are the loops that would have been ignored had the face's owning body been facetted, thus small holes in the face will not be ignored if the loop defining the hole is adjacent to faces that are not classed as small. In

this case the overall box will be the union of all of the 'topols' owning bodies boxes (with 'topol_transfs' applied).

'PK_facet_ignore_scope_local_c':
When looking for ignorable features the faces in the list of faces will be facetted as separate entities, hence there will be no matching between adjacent faces. If the face contains holes defined by loops which can be classed as small these will be ignored. In this case the overall box will be the union of all the 'topols' boxes (with 'topol_transfs' applied).

WIRE EDGES
The behaviour of the facetting operation on faces which contain wire edges can be modified via the 'wire_edges' option.

The permitted values for this option:

'PK_facet_wire_edges_no_c': (default)
Facetting will ignore wire edges in faces, they will be facetted over.

'PK_facet_wire_edges_yes_c':
Facetting will take wire edges in faces into account and will facet the face in such a way that no facet is intersected by a wire edge.

PK_TOPOL_facet_o_t

```
struct PK_TOPOL_facet_o_s
    {
    PK_TOPOL_facet_mesh_o_t       control;
    PK_TOPOL_facet_choice_o_t      choice;
    };
typedef struct PK_TOPOL_facet_o_s PK_TOPOL_facet_o_t;
```

Definition of PK_TOPOL_facet options structure.

Options are divided into :

- controls generate the facetted representation
- options which select which data are returned in tabular form

PK_TOPOL_facet_r_t

```
struct PK_TOPOL_facet_r_s
    {
    int number_of_facets;
    int number_of_fins;
    int number_of_vertices;
    PK_TOPOL_fctab_facet_fin_t      facet_fin;
    PK_TOPOL_fctab_fin_fin_t        fin_fin;
    PK_TOPOL_fctab_fin_vertex_t     fin_vertex;
    PK_TOPOL_fctab_vertex_point_t   vertex_point;
    PK_TOPOL_fctab_vertex_normal_t  vertex_normal;
    PK_TOPOL_fctab_vertex_param_t   vertex_param;
    PK_TOPOL_fctab_point_vec_t      point_vec;
    PK_TOPOL_fctab_normal_vec_t     normal_vec;
    PK_TOPOL_fctab_param_uv_t       param_uv;
    PK_TOPOL_fctab_param_dp_t       param_dp;
    PK_TOPOL_fctab_param_d2p_t      param_d2p;
    PK_TOPOL_fctab_facet_face_t     facet_face;
    PK_TOPOL_fctab_facet_occ_t      facet_occ;
    PK_TOPOL_fctab_edge_fin_t       edge_fin;
```

```
    PK_TOPOL_fctab_edge_occ_t       edge_occ;
    PK_TOPOL_fctab_error_facet_t    error_facet;
    };
typedef struct PK_TOPOL_facet_r_s PK_TOPOL_facet_r_t;
```

The PK_TOPOL_facet_r_t structure returns the total number of facets, fins and vertices in the facet mesh, with tables representing facet topology, geometry, associative and error data as requested in the PK_TOPOL_facet_choice_o_t options structure.

The returned structure always calculates and returns the number of facets, fins and vertices in the facet mesh, irrespective of choices selected by the PK_TOPOL_facet_choice_o_t options structure.

Each table is named in the form PK_TOPOL_fctab_<table>_t <table>
where each "<table>" field name corresponds to one of the PK_LOGICAL_t switch names in PK_TOPOL_facet_choice_o_t

If a specific table was requested (by setting the corresponding field name in PK_TOPOL_facet_choice_o_t to PK_LOGICAL_true), the table is returned with :

- an array pointer
- an integer which contains the number of array items

The array pointer is an address in virtual memory, provided by the memory allocator (as registered with PK_MEMORY_register_callbacks). All such virtual memory data must be freed by the application.

If a table was not requested, the table is returned with :
- an array pointer which contains NULL
- an integer length of zero

To free all of the memory allocated for the return tables, the function PK_TOPOL_facet_r_f should be called.

PK_TOPOL_fctab_edge_fin_t

```
struct PK_TOPOL_fcstr_edge_fin_s
       {
       int  edge;
       int   fin;
       };
typedef struct PK_TOPOL_fcstr_edge_fin_s
              PK_TOPOL_fcstr_edge_fin_t;
struct PK_TOPOL_fctab_edge_fin_s
       {
       PK_TOPOL_fcstr_edge_fin_t   *data;
       int                        length;
       };
typedef struct PK_TOPOL_fctab_edge_fin_s
              PK_TOPOL_fctab_edge_fin_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It represents a two column table which maps fin indices into edge entity values. (i.e. it defines which fin is derived from which PK edge entity). Note that the mapping from fin to edge is a sparse relation, where interior fins have no association with PK edges. The two column table provides a compact notation for such data.

PK_TOPOL_fctab_edge_occ_t

```
struct PK_TOPOL_fcstr_edge_occ_s
       {
       int  edge;
       int  occ;
       };
typedef struct PK_TOPOL_fcstr_edge_occ_s
               PK_TOPOL_fcstr_edge_occ_t;
struct PK_TOPOL_fctab_edge_occ_s
       {
       PK_TOPOL_fcstr_edge_occ_t  *data;
       int                        length;
       };
typedef struct PK_TOPOL_fctab_edge_occ_s
               PK_TOPOL_fctab_edge_occ_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It is provided for future expansion. It is returned empty.

PK_TOPOL_fctab_error_facet_t

```
struct PK_TOPOL_fctab_error_facet_s
       {
       int  *facet;
       int length;
       };
typedef struct PK_TOPOL_fctab_error_facet_s
               PK_TOPOL_fctab_error_facet_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It contains pairs of integers which indicate whether any faults were detected during facetting. If the table is returned empty, this denotes that no faults were found.

The first entry is an integer token of type PK_facet_fault_t.
The second is its associated facet index or a PK entity value.

first integer value in pair	second integer value in pair
PK_facet_fault_face_c	The specified PK_FACE_t entity could not be facetted using the specified mesh controls. It implies that part of the generated facet mesh is missing.
PK_facet_fault_facet_c	The specified facet (whose index is given) is disconnected from some neighbouring facets. The topology of the generated facet mesh is valid but may contain superflous 'slit edges'.
PK_facet_fault_vertex_point_c	The coordinate of the specified vertex (whose index is given) did not match any vertex_point coordinates on adjacent faces. The facet mesh may contain small gaps close to this point.
PK_facet_fault_vertex_normal_c	The facet function was unable to calculate an accurate surface normal value at the position of the facet vertex (whose index is given). The vector referenced by this vertex's 'vertex_normal' table should not be used.
PK_facet_fault_vertex_param_c	The facet function was unable to calculate accurate parameter values at the position of the facet vertex (whose index is given). The uv, dp and/or the dp2 parameter values

	referenced by this vertex's 'vertex_param' table should not be used.
PK_facet_fault_thin_face_c	No facets were output for the specified PK_FACE_t entity because the face is very thin. Although a facet may not be output for a given face, this is not an indication that facetting has failed.
PK_facet_fault_body_c	The specified PK_BODY_t entity could not be facetted using the specified mesh controls. It implies that no facet mesh has been generated for this body.

PK_TOPOL_fctab_facet_face_t

```
struct PK_TOPOL_fctab_facet_face_s
       {
       int   *face;
       int   length;
       };
typedef struct PK_TOPOL_fctab_facet_face_s
               PK_TOPOL_fctab_facet_face_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It represents a one column table which maps facet indices into face entity values (i.e. it defines which facet is derived from which PK face entity).

PK_TOPOL_fctab_facet_fin_t

```
struct  PK_TOPOL_fcstr_facet_fin_s
       {
       int     facet;
       int       fin;
       };
typedef struct PK_TOPOL_fcstr_facet_fin_s
               PK_TOPOL_fcstr_facet_fin_t;
struct PK_TOPOL_fctab_facet_fin_s
       {
       PK_TOPOL_fcstr_facet_fin_t *data;
       int   length;
       };
typedef struct PK_TOPOL_fctab_facet_fin_s
               PK_TOPOL_fctab_facet_fin_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It represents a two column table which 'groups' sets of fins owned by the same facet together.

PK_TOPOL_fctab_facet_occ_t

```
struct PK_TOPOL_fctab_facet_occ_s
       {
       int    *occ;
       int  length;
       };
typedef struct PK_TOPOL_fctab_facet_occ_s
               PK_TOPOL_fctab_facet_occ_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It is provided for future expansion. It is returned empty.

PK_TOPOL_fctab_fin_fin_t

```
struct PK_TOPOL_fctab_fin_fin_s
        {
        int     *fin;
        int   length;
        };
typedef struct PK_TOPOL_fctab_fin_fin_s
              PK_TOPOL_fctab_fin_fin_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It represents a one column table which maps fin indices onto co_fins. Fins without co_fins (laminar edges on sheet bodies or edges that lie on loops that are being ignored on one face) have their co_fin value set to -1. Fins with a single co-fin share the same start & end vertices as their co-fin but in reversed order. This mapping is commutative.

PK_TOPOL_fctab_fin_vertex_t

```
struct PK_TOPOL_fctab_fin_vertex_s
        {
        int  *vertex;
        int   length;
        };
 typedef struct PK_TOPOL_fctab_fin_vertex_s
               PK_TOPOL_fctab_fin_vertex_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It represents a one column table which maps fin indices into 'head' vertex indices.

PK_TOPOL_fctab_normal_vec_t

```
struct PK_TOPOL_fctab_normal_vec_s
       {
       PK_VECTOR_t *vec;
       int       length;
       };
typedef struct PK_TOPOL_fctab_normal_vec_s  PK_TOPOL_fctab_normal_vec_t;
```

This is a support structure for PK_TOPOL_facet_r_t.
It represents a one column table of vertex normals.

PK_TOPOL_fctab_param_d2p_t

```
struct PK_TOPOL_fcstr_param_d2p_s
       {
       PK_VECTOR_t  du2;
       PK_VECTOR_t dudv;
       PK_VECTOR_t  dv2;
       };
typedef struct PK_TOPOL_fcstr_param_d2p_s
              PK_TOPOL_fcstr_param_d2p_t;
struct PK_TOPOL_fctab_param_d2p_s
       {
       PK_TOPOL_fcstr_param_d2p_t *data;
       int                        length;
       };
typedef struct PK_TOPOL_fctab_param_d2p_s
              PK_TOPOL_fctab_param_d2p_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It represents a three column table of d2p/du2, d2p/dudv & d2p/dv2 second derivative surface values

PK_TOPOL_fctab_param_dp_t

```
struct PK_TOPOL_fcstr_param_dp_s
       {
       PK_VECTOR_t du;
       PK_VECTOR_t dv;
       };
typedef struct PK_TOPOL_fcstr_param_dp_s
               PK_TOPOL_fcstr_param_dp_t;
struct PK_TOPOL_fctab_param_dp_s
       {
       PK_TOPOL_fcstr_param_dp_t *data;
       int                       length;
       };
typedef struct PK_TOPOL_fctab_param_dp_s  PK_TOPOL_fctab_param_dp_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It represents a two column table of du and dv first derivative surface values

PK_TOPOL_fctab_param_uv_t

```
struct PK_TOPOL_fcstr_param_uv_s
       {
       double u;
       double v;
       };
typedef struct PK_TOPOL_fcstr_param_uv_s
               PK_TOPOL_fcstr_param_uv_t;
struct PK_TOPOL_fctab_param_uv_s
       {
       PK_TOPOL_fcstr_param_uv_t *data;
       int                       length;
       };
typedef struct PK_TOPOL_fctab_param_uv_s  PK_TOPOL_fctab_param_uv_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It represents a two column table of u,v surface parameterisation values

PK_TOPOL_fctab_point_vec_t

```
struct PK_TOPOL_fctab_point_vec_s
       {
       PK_VECTOR_t *vec;
       int       length;
       };
typedef struct PK_TOPOL_fctab_point_vec_s  PK_TOPOL_fctab_point_vec_t;
```

This is a support structure for PK_TOPOL_facet_r_t.
It represents a one column table of vertex coordinates.

PK_TOPOL_fctab_vertex_normal_t

```
struct PK_TOPOL_fctab_vertex_normal_s
       {
       int        *normal;
       int        length;
       };
typedef struct PK_TOPOL_fctab_vertex_normal_s
              PK_TOPOL_fctab_vertex_normal_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It structure represents a one column table which maps vertex indices (topology) into a table of 'normal' indices.

PK_TOPOL_fctab_vertex_param_t

```
struct PK_TOPOL_fctab_vertex_param_s
       {
       int        *param;
       int        length;
       };
typedef struct PK_TOPOL_fctab_vertex_param_s
              PK_TOPOL_fctab_vertex_param_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It represents a one column table which maps vertex indices (topology) into a parameter index value. This index can be used to reference the param_uv, param_dp and param_d2p tables.

PK_TOPOL_fctab_vertex_point_t

```
struct PK_TOPOL_fctab_vertex_point_s
       {
       int        *point;
       int        length;
       };
typedef struct PK_TOPOL_fctab_vertex_point_s
              PK_TOPOL_fctab_vertex_point_t;
```

This is a support structure for PK_TOPOL_facet_r_t. It represents a one column table which maps vertex indices (topology) into a table of 'point' indices (geometry).

PK_TOPOL_local_r_t

```
struct PK_TOPOL_local_r_s
    {
    PK_local_status_t    status;
    int                  n_error_entities;
    PK_ENTITY_t         *error_entities;
    };
typedef struct PK_TOPOL_local_r_s PK_TOPOL_local_r_t;
```

This structure is used to return the status of the local operation and in the case of a failure, any associated entities.

When a call to the local operation returns a value of PK_ERROR_no_errors, then the actual status of the operation is given by the 'status' field.

If a fault is indicated by the 'status' field, then the number of entities associated with the fault are given by the field 'n_error_entities', and the entities are returned in the array 'error_entities'.

Description of fields:

'status' This field indicates the overall state of the local operation. See PK_local_status_t for values and meanings.

'n_error_entities' The number of topological or geometric entities in the 'error_entities' array.

'error_entities' If the 'status' field of this return structure indicates a failure, this array may contain topological or geometric entities which are associated with the failure.

Error entities returned:
The 'error_entities' array will contain the following entities according to the value of the 'status' field :

PK_local_status_ok_c	: No entities		
PK_local_status_nocheck_c	: No entities		
PK_local_status_fail_c	: No entities		
PK_local_status_cant_get_pt_c	: 1	VERTEX	Failed to find new geometry for existing vertex.
	n	FACEs	Failed to find geometry for new vertex.
PK_local_status_cant_get_cu_c	: 1	EDGE	Failed to find new geometry for existing edge.
	n	FACEs	Failed to find geometry for new edge.
PK_local_status_side_cu_fail_c	: 1	EDGE	Failed to find side curve.
PK_local_status_side_su_fail_c	: 1	FACE	Failed to create a side surface.
PK_local_status_cant_get_su_c	: n	FACEs	Unable to produce offset surfaces for these faces. No surface replacement is performed.
PK_local_status_cant_offset_c	: 1	FACE	Failed to offset or face could not be deleted.
PK_local_status_fa_fail_c	: 1	FACE	Face which failed check.
PK_local_status_fa_fa_fail_c	: 2	FACEs	Pair of faces where inconsistency found.
PK_local_status_ed_remains_c	: 1	EDGE	An existing edge should have disappeared
	n	FACEs	A transient edge should have disappeared
PK_local_status_point_contact_c	: 1	VERTEX	Vertex where point contact has has been found.
PK_local_status_not_supported_c	: 1	FACE	Face that has currently unsupported combination of reference entities.

Any space allocated by Parasolid within this structure may be freed by calling PK_TOPOL_local_r_f.

PK_TOPOL_range_array_o_t

```
struct PK_TOPOL_range_array_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    PK_range_bound_t    bound;          --- bounds on min separation
    };

typedef struct PK_TOPOL_range_array_o_s PK_TOPOL_range_array_o_t;
```

Holds optional controls for finding the global minimum separation between two arrays of topological entities.

The option structure defines options applicable to finding the global minimum separation between two arrays of topological entities.

PK_TOPOL_range_array_vector_o_t

```
struct PK_TOPOL_range_array_vector_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    PK_range_bound_t    bound;          --- bounds on min separation
    };

typedef struct PK_TOPOL_range_array_vector_o_s PK_TOPOL_range_array_vector_o_t;
```

Holds optional controls for finding the global minimum separation between an array of topological entities and a position.

The option structure defines options applicable to finding the global minimum separation between an array of topological entities and a position.

PK_TOPOL_range_local_o_t

```
struct PK_TOPOL_range_local_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    };

typedef struct PK_TOPOL_range_local_o_s PK_TOPOL_range_local_o_t;
```

Holds optional controls for finding the local minimum separations between two topological entities.

The option structure defines options applicable to finding the local minimum separations between two topological entities.

PK_TOPOL_range_local_vector_o_t

```
struct PK_TOPOL_range_local_vector_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    };

typedef struct PK_TOPOL_range_local_vector_o_s PK_TOPOL_range_local_vector_o_t;
```

Holds optional controls for finding the local minimum separations between a topological entity and a position.

The option structure defines options applicable to finding the local minimum separations between a topological entity and a position.

PK_TOPOL_range_o_t

```
struct PK_TOPOL_range_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    PK_range_bound_t    bound;          --- bounds on min separation
    PK_range_guess_t    guesses[2];     --- guesses for each end
    };

typedef struct PK_TOPOL_range_o_s PK_TOPOL_range_o_t;
```

Holds optional controls for finding the global minimum separation between two topological entities.

The option structure defines options applicable to finding the global minimum separation between two topological entities.

PK_TOPOL_range_vector_o_t

```
struct PK_TOPOL_range_vector_o_s
    {
    int                 o_t_version;    --- version number of option structure
    PK_LOGICAL_t        have_tolerance; --- (PK_LOGICAL_false)
    double              tolerance;      --- tolerance
    PK_range_bound_t    bound;          --- bounds on min separation
    PK_range_guess_t    guess;          --- guess for end on 'topol'
    };

typedef struct PK_TOPOL_range_vector_o_s PK_TOPOL_range_vector_o_t;
```

Holds optional controls for finding the global minimum separation between a topological entity and a position.

The option structure defines options applicable to finding the global minimum separation between a topological entity and a position.

PK_TOPOL_render_facet_go_o_t

```
struct PK_TOPOL_render_facet_go_o_s
    {
    int                        o_t_version;   --- version of PK option struct
    PK_facet_go_normals_t      go_normals;    --- output GO surface normals
                                              --- (PK_facet_go_normals_no_c)
    PK_facet_go_parameters_t   go_parameters; --- output GO surf parameters
                                              --- (PK_facet_go_parameters_no_c)
    PK_facet_go_edges_t        go_edges;      --- output GO edge entities
                                              --- (PK_facet_go_edges_no_c)
    PK_facet_go_strips_t       go_strips;     --- output facets in strips
                                              --- (PK_facet_go_strips_no_c)
    int                        go_max_facets_per_strip;
                                              --- associated max value (0)
    PK_facet_go_interleaved_t go_interleaved;--- interleave the output of
                                              --- faces from different bodies
                                              --- (PK_facet_go_interleaved_no_c)
    };
typedef struct PK_TOPOL_render_facet_go_o_s PK_TOPOL_render_facet_go_o_t;
```

Selection of graphics output (GO) options for PK_TOPOL_render_facet_o_t

Specific Errors

```
    PK_ERROR_bad_value              PK_facet_go_strips_yes_c needs value > 1
```

```
  This selects which additional data are output through GO

    - surface normals
    - surface parameters
    - model edge entities

  and how these are output.

  Facets may be output in one of two ways: all facets for a particular body are
  output before facets belonging to any other body are output, or, the facets
  of faces of the bodies being facetted can be interleaved.  See documentation
  of 'PK_topol_facet_go_interleaved_t' for details.
```

PK_TOPOL_render_facet_o_t

```
struct PK_TOPOL_render_facet_o_s
    {
    PK_TOPOL_facet_mesh_o_t       control;
    PK_TOPOL_render_facet_go_o_t  go_option;
    };
typedef struct PK_TOPOL_render_facet_o_s PK_TOPOL_render_facet_o_t;
```

Definition of PK_TOPOL_render_facet options structure.

```
Options are divided into :

  - controls generate the facetted representation
  - options which select those data which are output through GO
```

PK_TOPOL_render_line_o_t

```
struct PK_TOPOL_render_line_o_s
    {
    int                       o_t_version;--- version of this PK option struct
    PK_render_edge_t          edge;       --- whether to draw edges
                                          --- (PK_render_edge_yes_c)
    PK_render_silhouette_t    silhouette; --- whether to draw silhouettes
                                          --- (PK_render_silhouette_no_c)
    PK_render_planar_t        planar;     --- whether to draw planar hatching
                                          --- (PK_render_planar_no_c)
    PK_render_radial_t        radial;     --- whether to draw radial hatching
                                          --- (PK_render_radial_no_c)
    PK_render_param_t         param;      --- whether to draw parametric hatch
                                          --- (PK_render_param_no_c)
    PK_render_unfix_t         unfix;      --- whether to draw unfixed blends
                                          --- (PK_render_unfix_no_c)
    PK_render_vis_t           visibility; --- whether to determine visibility
                                          --- (PK_render_vis_no_c)
    PK_render_smooth_t        smooth;     --- whether to indicate smooth edges
                                          --- (i.e. where faces have the same
                                          ---     tangent surface at the edge)
                                          --- (PK_render_smooth_no_c)
    PK_render_internal_t      internal;   --- whether to indicate internal edges
                                          --- (i.e. where faces on either side
                                          ---  of the edge lie on same surface)
                                          --- (PK_render_internal_no_c)
    PK_render_region_t        region;     --- whether to produce regional data
                                          --- (PK_render_region_no_c)
    PK_render_hierarch_t      hierarch;   --- whether to produce hierarchical
                                          --- output (PK_render_hierarch_no_c)
    PK_render_bcurve_t        bcurve;     --- method for rendering bcurves
                                          --- (PK_render_bcurve_polyline_c)
    PK_render_viewport_t      viewport;   --- whether to use viewport(s)
                                          --- (PK_render_viewport_no_c)
    PK_render_invisible_t     invisible;  --- whether to output invisible lines
                                          --- when 'visibility' is set to
                                          --- PK_render_vis_extended_c
                                          --- (PK_render_drafting_no_c)
    PK_render_drafting_t      drafting;   --- whether to distinguish lines
                                          --- hidden by lines and faces
                                          --- when 'visibility' is set to
                                          --- PK_render_vis_extended_c
                                          --- (PK_render_invisible_no_c)
    PK_render_self_hidden_t   self_hidden;--- whether to distinguish lines
                                          --- hidden by own or another
                                          --- occurrence when 'visibility' is
                                          --- set to PK_render_vis_extended_c
                                          --- (PK_render_self_hidden_no_c)
    PK_LOGICAL_t              is_curve_chord_tol;
                                          --- whether curve_chord_tol is set
                                          --- ( PK_LOGICAL_false )
    double                    curve_chord_tol;
                                          --- chordal tolerance between curve
                                          --- and chord edges in model units
                                          --- ( 0.0 )
    PK_LOGICAL_t              is_curve_chord_max;
                                          --- whether curve_chord_max is set
                                          --- ( PK_LOGICAL_false )
    double                    curve_chord_max;
                                          --- max length of chord in model units
                                          --- ( 0.0 )
    PK_LOGICAL_t              is_curve_chord_ang;
                                          --- whether curve_chord_ang is set
                                          --- ( PK_LOGICAL_false )
    double                    curve_chord_ang;
                                          --- max angle between adjacent chords
```

```
                                        ---                         in radians
                                        --- ( 0.0 )
double                  planar_spacing;
                                        --- distance between planes for
                                        --- planar hatching
                                        --- ( 0.0 )
PK_AXIS1_sf_t           planar_axis;
                                        --- axis for planar hatching
                                        --- (through 0,0,0 direction 0,0,0)
double                  radial_around;
                                        --- angular spacing around spine for
                                        --- radial hatching
                                        --- ( 0.0 )
double                  radial_along;
                                        --- spacing in model units along spine
                                        --- ( 0.0 )
double                  radial_about;
                                        --- angular spacing about centre
                                        --- ( 0.0 )
double                  param_u;
                                        --- hatch spacing in U direction
                                        --- ( 0.0 )
double                  param_v;
                                        --- hatch spacing in V direction
                                        --- ( 0.0 )
double                  unfix_spacing;
                                        --- unfixed blend default rib spacing
                                        --- ( 0.0 )
PK_NABOX_sf_t           viewport_sf;
                                        --- viewport definition
                                        --- if single viewport is used, i.e.
                                        --- PK_render_viewport_yes_c chosen
double                  radial_around_start;
                                        --- starting value for angular
                                        --- spacing around
                                        --- spine for radial hatching
                                        --- ( 0.0 )
double                  radial_along_start;
                                        --- starting value for spacing in
                                        --- model units along spine
                                        --- ( 0.0 )
double                  radial_about_start;
                                        --- starting value for angular
                                        --- spacing about centre
                                        --- ( 0.0 )
double                  param_u_start;
                                        --- starting value for hatch spacing
                                        --- in U direction
                                        --- ( 0.0 )
double                  param_v_start;
                                        --- starting value for hatch spacing
                                        --- in V direction
                                        --- ( 0.0 )
PK_render_ignore_t      ignore;
                                        --- whether to ignore small features
                                        --- in hidden line rendering.
                                        --- Small features can be ignored
                                        --- by inputting an absolute value or
                                        --- a ratio value
                                        --- ( PK_render_ignore_no_c )
double                  ignore_value;
                                        --- features smaller than this value
                                        --- will be considered for ignoring
                                        --- ( 0.0 )
PK_render_overlap_t     overlap;
                                        --- whether to allow for overlapping
                                        --- bodies
```

```
                                        --- ( PK_render_overlap_no_c )
    PK_render_memory_target_t   memory_target;
                                        --- whether to restrict memory usage
                                        --- ( PK_render_memory_target_no_c )
    size_t                      memory_target_value;
                                        --- target memory usage in bytes
                                        --- ( 0 )
    PK_LOGICAL_t                is_edge_smooth_tol;
                                        --- whether edge_smooth_tol is set
                                        --- ( PK_LOGICAL_false )
    double                      edge_smooth_tol;
                                        --- max angle (in radians) allowed
                                        --- between face
                                        --- normals for edge to be smooth
                                        --- ( 0.0 )
    int                         n_viewports;
                                        --- number of viewports
    const PK_NABOX_sf_t     *viewports;
                                        --- array of viewports
    };
typedef struct PK_TOPOL_render_line_o_s PK_TOPOL_render_line_o_t;
```

Controls for the PK_TOPOL_render_line options structure

PK_TOPOL_render_line_o_t provides controls for PK_TOPOL_render_line to generate view-independent, view-dependent and hidden line drawings of topological entities through the Graphics Output (GO) interface.

The controls allow view independent and view dependent data (silhouettes) to be generated in the same call to PK_TOPOL_render_line. Hidden line drawings should be generated in a separate call to PK_TOPOL_render_line because their arguments are generally incompatible with those used for the other categories of drawing.

The options specify which types of lines should be rendered (e.g. edges, silhouettes) and what additional data (e.g. visibility and smoothness) needs to be calculated and handed onto GO.

Tolerance Values
Each control tolerance is passed as a pair of values, consisting of a PK_LOGICAL_t flag and a double precision value. If a specific tolerance is required, the variable is set to a strictly positive value and the associated flag set to PK_LOGICAL_true.

If Parasolid is to use its internal tolerance defaults, the associated flag is set to PK_LOGICAL_false (and no use made of the associated tolerance argument). This is the default state set by the PK_TOPOL_render_line_o_m macro.

Visibility
If 'visibility' is set to PK_render_vis_no_c then no visibility evaluation (ie hidden line drawing) is performed. Common visibility requirements may be requested by the values: PK_render_vis_hid_c, PK_render_vis_inv_c and PK_render_vis_inv_draft_c. For more refined control over visibility requirements, 'visibility' may be set to PK_render_vis_extended_c which then enables the options: 'invisible', 'drafting' and 'self_hidden'.

Radial and Param Default Start Values
In no values are specified for the radial start values then the option PK_render_radial_yes_c will behave like PK_render_radial_free_c and Parasolid will choose the start values. Similarly, if no param start values are specified then the option PK_render_param_yes_c will behave like PK_render_param_free_c and Parasolid will choose the start values.

Ignore Small Features
The 'ignore' option can be set to one of the following values:

PK_render_ignore_no_c: (default) All features will be rendered regardless of their size.

PK_render_ignore_absolute_c: Any feature with a box size smaller than the value specified will be ignored.

PK_render_ignore_ratio_c: The 'ignore_value' corresponds to the ratio of box size of the feature to the size of the box containing the entities to be rendered i.e. if this value is set to 0.01 then all features smaller than 1 per-cent of the entities' box will be ignored.

PK_render_ignore_body_ratio_c: The 'ignore_value' corresponds to the ratio of the box size of the feature to the size of the box of the body containing the feature.

This option only takes effect when visibility is not set to PK_render_vis_no_c. Note that 'small' features will only be ignored provided that ignoring them does not leave the model invalid.

Overlapping Bodies
The 'overlap' options can be set to one of the following values:

PK_render_overlap_no_c: Parasolid assumes that the bodies to be rendered do not overlap. If there are overlapping entities present then the overlap will not be detect and the resulting picture may contain errors.

PK_render_overlap_yes_c: Parasolid will detect overlapping bodies and will split the existing lines appropriately. However, the algorithm will not generate new edges where the faces intersect. The additional effort of detecting overlaps will incur a performance overhead compared to 'PK_render_overlap_no_c'.

This option only takes effect when visibility is not set to PK_render_vis_no_c. Abutting bodies will always be rendered correctly, regardless of the setting of the option.

Memory Target
The 'memory target' option can be set to one of the following values:

PK_render_memory_target_yes_c: Parasolid will attempt to restrict the memory usage of PK_TOPOL_render_line to the amount of memory given by "memory_target_value". The target value is not guaranteed to be met. The amount of memory used by PK_TOPOL_render_line will be reduced at the expense of total processor time.

The value given by 'memory_target' refers to the peak amount of extra memory temporarily allocated during PK_TOPOL_render_line.

PK_render_memory_target_no_c: Parasolid will maximise performance without attempting to restrict memory usage.

This option only takes effect when visibility is not set to PK_render_vis_no_c.

Edge Smoothness
If 'is_edge_smooth_tol' is set to true, the value in 'edge_smooth_tol' defines the angular tolerance (in radians) for the smoothness property in the same way as PK_EDGE_is_smooth.

The criteria for smoothness is defined as being that the angle between the normals of the two faces that meet at a given edge do not exceed the angle supplied in 'edge_smooth_tol'.

If 'is_edge_smooth_tol' is set to false, the angular tolerance used will depend upon the accuracy of the relevant edge.

Multiple viewports
PK_render_viewport_t controls the multiple viewport behaviour.
PK_render_viewport_no_c: no viewports are supplied (default)

PK_render_viewport_yes_c: the single viewport specified in 'viewport_sf' is used. Entities outside the single supplied viewport may be omitted. In this case, 'n_viewports' and 'viewports' are ignored.

PK_render_viewport_array_c: an array of viewports is supplied. Entities outside all the supplied viewports may be omitted. 'viewport_sf' is ignored. The number of viewports used is set in 'n_viewports' and can be set to zero or a positive integer. The array of viewports is specified in 'viewports'.

PK_TOPOL_sense_t

```
typedef int PK_TOPOL_sense_t;
```

PK_TOPOL_t

This type represents any TOPOLogical item.

A topological item is a type of entity (PK_CLASS_topol is a subclass of PK_CLASS_entity).

A topological item may be of one of the types:

- PK_PART_t: a part which may be saved or received
- PK_INSTANCE_t: an instance which positions and orientates an occurrence of a part
- PK_REGION_t: a region which is a connected subset of 3-dimensional space
- PK_SHELL_t: a shell which is a connected set of oriented faces
- PK_FACE_t: a face which is a bounded connected subset of a surface
- PK_EDGE_t: an edge which is a bounded connected subset of a curve
- PK_FIN_t: a fin which is the junction between a face and an edge
- PK_LOOP_t: a loop which is a closed connected set of fins forming a face boundary.
- PK_VERTEX_t: a vertex which is a single position in 3D space

PK_TOPOL_track_r_t

```
struct PK_TOPOL_track_r_s
    {
    int                          n_track_records;
    PK_TOPOL_track_record_r_t  *track_records;

    int                         *internal_origs;
    PK_CLASS_t                  *internal_classes;
    int                         *internal_prods;
    };
typedef struct PK_TOPOL_track_r_s PK_TOPOL_track_r_t;
```

This data structure contains all the generic topology tracking records for the modelling operation.

Tracking information will be returned via generic tracking structures. Each returned sub-structure will record one item of tracking information. The structure will associate one group of "originating" topology with another group of "product" topology for a particular tracking "event". Originating topology will have the class types recorded seperately as in some cases these entities will have been deleted and their identifiers will refer to a non-existent entity.

Description of fields:

'n_track_records' The length of the 'track_records' array.

'track_records' The array of tracking information records.

The following fields are for internal use only and are not to be accessed by the application.

'internal_origs' Internal use only

'internal_classes' Internal use only

'internal_prods' Internal use only

Destroying the returned data structure

When the application wishes to free the memory associated with the tracking structure, the function 'PK_TOPOL_track_r_f' must be used.

PK_TOPOL_track_record_r_t

```
struct PK_TOPOL_track_record_r_s
    {
    PK_TOPOL_track_t           track;
    int                        n_original_topols;
    int                       *original_topols;
    PK_CLASS_t                *original_classes;
    int                        n_product_topols;
    PK_TOPOL_t                *product_topols;
    };
typedef struct PK_TOPOL_track_record_r_s PK_TOPOL_track_record_r_t;
```

This data structure encapsulates the association between topologies before and after the modelling operation.

Description of fields:

'track' This value of this field indicates what type of tracking information the record represents. Permitted values are :

PK_TOPOL_track_delete_c
PK_TOPOL_track_merge_c
PK_TOPOL_track_create_c
PK_TOPOL_track_split_c
PK_TOPOL_track_derive_c

'n_original_topols' The length of the 'original_topols' and 'orig_classes' arrays.

'original_topols' The participating topologies in the modelling operation. The class of the entity is held in the corresponding element of 'original_classes' array.

'original_classes' The class of each participating topology within the in the 'original_topols' array.

'n_product_topols' The length of the 'product_topols' array.

'product_topols' The topologies which are a resultant of the modelling operation.

The Contents of the Structure

The expectations on the contents of the structure for each track type :

PK_TOPOL_track_delete_c

- o 'product_topols' will be empty.
- o The class of the deleted entities can be determined from the corresponding entry in the 'original_classes' array.
- o 'original_topols' contains some or all of the deleted topologies.

PK_TOPOL_track_merge_c

- o 'original_topols' contains the list of topologies merged.
- o The class of the merged entities can be determined from the corresponding entry in the 'original_classes' array.
- o 'product_topols' holds the list of surviving topology.

PK_TOPOL_track_create_c

- o 'original_topols' holds the list of topology copied, or the topology from which the existence of the products was inferred.
- o The class of the original entities can be determined from the corresponding entry in the 'original_classes' array.
- o 'product_topols' holds the newly created topology.

PK_TOPOL_track_split_c

- o 'original_topols' holds the list of topology that was split.
- o The class of the original entities can be determined from the corresponding entry in the 'original_classes' array.
- o 'product_topols' holds the list of topology resulting from the splitting operation.

PK_TOPOL_track_t

typedef int PK_TOPOL_track_t;

This data type indicates what type of tracking data is being output. It has four values:

PK_TOPOL_track_delete_c	marks deletion of topologies
PK_TOPOL_track_merge_c	marks deletion of topology by merging
PK_TOPOL_track_create_c	marks creation of topology from scratch or by copying
PK_TOPOL_track_split_c	marks creation of topology by sub-division
PK_TOPOL_track_derive_c	marks creation of topology by combination of merging and splitting

PK_TORUS_array_t

This structure contains an array of entities of type PK_TORUS_t

```
typedef struct PK_TORUS_array_s
    {
    PK_TORUS_t     *array;   --- array of TORUSs
    int             length;  --- length of array
```

```
    }
    PK_TORUS_array_t;
```

It has the fields:

'array': array of entities of type PK_TORUS_t

'length': the number of entities in 'array'

PK_TORUS_sf_t

```
struct PK_TORUS_sf_s
    {
    PK_AXIS2_sf_t basis_set;    --- centre, axis and a reference direction
    double        major_radius; --- the major radius of the torus(!=0)
    double        minor_radius; --- the minor radius of the torus(>0)
    };

typedef struct PK_TORUS_sf_s PK_TORUS_sf_t;
```

Specific Errors

```
PK_ERROR_radius_sum_le_0        sum of radii less than or equal to zero
PK_ERROR_radius_le_0           'minor_radius' less than or equal to zero
PK_ERROR_radius_eq_0           'major_radius' equals zero
```

The parameterisation of the torus is as follows:

```
    C = basis_set.location
    x = basis_set.ref_direction
    y = basis_set.axis X basis_set.ref_direction ( cross product )
    z = basis_set.axis
    R = major_radius
    r = minor_radius

    S(u, v) = C + (R + r(cos v))( ( cos u )x + ( sin u )y ) + r( sin v )z

                                where  ( 0 <= u < 2PI; -PI <= v < PI )
```

For certain choices of 'major_radius' and 'minor_radius' the torus will self-intersect. There are two cases for self-intersecting surfaces.

The first case produces an inner and an outer surface. The inner surface is called a lemon, while the outer surface is called an apple. Each surface is bounded by two singular points, where the normal is undefined, and treated as a separate surface.

The second case produces an outer surface only. This apple has only one singular point, on its axis of rotation.

An apple is created by setting 'major_radius' positive and less than or equal to 'minor_radius'. If, on the other hand, 'major_radius' is negative and the sum of the radii is positive then a lemon is produced.

PK_TORUS_t

This type represents a TORUS.

A torus is a type of surface (PK_CLASS_torus is a subclass of PK_CLASS_surf).

The torus is defined by data contained in a PK_TORUS_sf_t.

PK_TRANSF_array_t

This structure contains an array of entities of type PK_TRANSF_t

```
typedef struct PK_TRANSF_array_s
    {
    PK_TRANSF_t    *array;   --- array of TRANSFormations
    int             length;  --- length of array
    }
    PK_TRANSF_array_t;
```

It has the fields:

'array': array of entities of type PK_TRANSF_t

'length': the number of entities in 'array'

PK_TRANSF_check_o_t

```
struct PK_TRANSF_check_o_s
    {
    int        o_t_version;    --- version number of option structure
    int        max_faults;     --- limit on fault numbers (1)
    };

typedef struct PK_TRANSF_check_o_s PK_TRANSF_check_o_t;
```

Holds optional controls for checking a transformation.

The option structure defines options applicable to checking a transformation.

PK_TRANSF_sf_t

```
struct PK_TRANSF_sf_s
    {
    double   matrix[4][4];   --- homogeneous transformation matrix
    };

typedef struct PK_TRANSF_sf_s PK_TRANSF_sf_t;
```

Specific Errors

```
PK_ERROR_wrong_transf            determinant is zero
PK_ERROR_sc_factor_le_0          scale must be greater than zero
```

The array 'matrix' contains the components that make up the transformation.

The matrix (M) operates as a post-multiplier on row vectors containing homogenous coordinates thus:

(x',y',z',s') = (x,y,z,s) M

where the conventional 3-d coordinates are

(x/s,y/s,z/s).

The matrix thus consists of

```
    (             , Px )
    (      R      , Py )
    (             , Pz )
    ( Tx, Ty, Tz,   S  )

R = a non singular transformation matrix.
    This matrix contains the rotation, reflection, non-uniform scaling
    and shearing components.
T = a translation vector.
P = perspective terms in viewing transformations.
    Must be zero in transformations used for modelling.
S = a global scaling factor. It has to be greater than zero.
    Its value is the inverse of the global scale.

The subscripts of 'matrix' corresponding to the above form are:

    ( [0][0]  [1][0]  [2][0]  [3][0] )
    ( [0][1]  [1][1]  [2][1]  [3][1] )
    ( [0][2]  [1][2]  [2][2]  [3][2] )
    ( [0][3]  [1][3]  [2][3]  [3][3] )

For a transformation involving uniform scale, it is advisable, although not
compulsory, that the magnitude of the determinant of R be one to Parasolid
angular tolerance (and so the uniform scale is 1/S). If the determinant is not
one then geometry types may change when transformed as explained under
PK_GEOM_transform.
```

PK_TRANSF_t

This type represents a TRANSForm. Transforms involving scale and/or rotation and/or translation are used in modelling operations and those which may also include perspective are used in viewing operations.

```
A transform is a type of entity (PK_CLASS_transf is a subclass of
PK_CLASS_entity).
```

PK_transmit_format_t

```
typedef int PK_transmit_format_t;

PK_transmit_format_t has the values:

    PK_transmit_format_text_c       text
    PK_transmit_format_binary_c     machine-dependent binary
    PK_transmit_format_neutral_c    machine-independent binary
    PK_transmit_format_applio_c     call registered output functions
    PK_transmit_format_xml_c        XML-compatible text (output only)
```

PK_TRCURVE_array_t

This structure contains an array of entities of type PK_TRCURVE_t

```
typedef struct PK_TRCURVE_array_s
    {
    PK_TRCURVE_t   *array;   --- array of TRimmed CURVEs
    int             length;  --- length of array
    }
    PK_TRCURVE_array_t;
```

It has the fields:

'array': array of entities of type PK_TRCURVE_t

'length': the number of entities in 'array'

PK_TRCURVE_sf_t

```
struct PK_TRCURVE_sf_s
    {
    PK_CURVE_t      basis_curve;  --- basis curve
    PK_INTERVAL_t   t_int;        --- interval on basis curve
    };

typedef struct PK_TRCURVE_sf_s PK_TRCURVE_sf_t;
```

This is the standard form of a trimmed curve.

PK_TRCURVE_t

This type represents an old form of TRimmed CURVE.

TRCURVE is not a fully supported curve type (PK_CLASS_trcurve is a subclass of PK_CLASS_geom). It allows access to the data for trimmed curves contained in old models.

The trcurve is defined by data contained in a PK_TRCURVE_sf_t

PK_UCHAR_t

```
typedef unsigned short PK_UCHAR_t;
```

This type represents a Unicode character.

PK_UCOPRD_f_t

```
typedef int  (*PK_UCOPRD_f_t)(const int  guise,    const int  format,
                              const PK_UCHAR_t name[],
                              const PK_LOGICAL_t skiphd,
                                    int *strid);
```

Open various guises of part file for reading

Receives:

int	guise	--- class of file: --- FFCSNP,FFCXMT --- FFCXMP,FFCXMD --- snapshot,C-transmit, --- partition,delta
int	format	--- format code: --- FFBNRY,FFTEXT (binary or text)
PK_UCHAR_t	name[]	--- key which identifies file --- (null-terminated Unicode)
PK_LOGICAL_t	skiphd	--- action required on opening file --- PK_LOGICAL_true: --- skip header (the usual case) or --- PK_LOGICAL_false:

```
                                                ---  leave header (used by Frustrum
                                                ---                acceptance tests)

Returns:
  int                        *strid             --- id for stream on which
                                                --- file is open
Return codes:                 FR_no_errors
                              FR_bad_name
                              FR_not_found
                              FR_bad_header
                              FR_open_fail

Description:
  This function opens various guises of existing part files for
  reading, i.e. C transmit, partition and deltas, and snapshot
  files, using a Unicode key.

  If the 'skiphd' flag is set to PK_LOGICAL_true, the header data is
  skipped when the file is opened. This is the mode which is used by
  Parasolid.

  If the 'skiphd' flag is set to PK_LOGICAL_false, the header data is not
  skipped when the file is opened; the preamble, the parts data and the
  trailer are read by subsequent calls to FFREAD. This mode is only used
  by the TESTFR to validate what has been written by a particular
  implementation.

  The routine returns a frustrum stream identifier or 'strid'.
  This is used in subsequent calls to FFREAD and FFCLOS.
```

PK_UCOPWR_f_t

```
typedef int  (*PK_UCOPWR_f_t)(const int  guise,    const int  format,
                              const PK_UCHAR_t name[],
                              const char pr2hdr[],
                                    int *strid);
```

Open various guises of part file for writing

```
Receives:
  int                         guise            --- class of file:
                                               ---  FFCSNP,FFCXMT,
                                               ---  FFCXMP,FFCXMD
                                               ---  snapshot,C-transmit,
                                               ---  partition,delta
  int                         format           --- format code:
                                               ---  FFBNRY,FFTEXT,FFXML
                                               ---  (binary,text,XML)
  PK_UCHAR_t                  name[]           --- key which identifies file
                                               ---  (null-terminated Unicode)
  char                        pr2hdr[]         --- part 2 header data from
                                               --- Parasolid (null terminated)

Returns:
  int                         strid            --- id for stream on which
                                               --- file is open

Return codes:                 FR_no_errors
                              FR_bad_name
                              FR_already_exists
                              FR_open_fail
                              FR_write_fail
                              FR_disc_full
```

Description:
  This function opens various guises of new files for writing, i.e.
  C transmit, snapshot, partition and delta files, using a Unicode key.

  The 'pr2hdr' string contains a sequence of keyword names and values in
  the form KEYWORD1=value1;KEYWORD2=value2 etc. These describe the
  implementation of Parasolid, for storage in the file header.
  The characters in the keyword values do not need to be escaped
  when they are written to file.

  The routine returns a frustrum stream identifier or 'strid'.
  This is used in subsequent calls to FFWRIT and FFCLOS.

PK_UV_t

```
struct PK_UV_s
    {
    double param[2];
    };
typedef struct PK_UV_s PK_UV_t;
```

This structure represents a position in a surface's parameter space.

The components are as follows:

    'param[0]' U parameter of surface
    'param[1]' V parameter of surface

PK_UVBOX_t

```
struct PK_UVBOX_s
    {
    double param[4];
    };
typedef struct PK_UVBOX_s PK_UVBOX_t;
```

This structure represents a rectangular region of a surface's parameter space.

Specific Errors

PK_ERROR_bad_uvbox	not a valid region of a surface's parameter space.

The doubles in a PK_UVBOX_t are in the following order:

    param[0] low U value
    param[1] low V value
    param[2] high U value
    param[3] high V value

'param[0]' must be less than or equal to 'param[2]'.
If these two values are equal, the uvbox has a zero extent in the U direction.
'param[1]' must be less than or equal to 'param[3]'.
If these two values are equal, the uvbox has a zero extent in the V direction.

The low U and V values both lie inside the surface's parameter range as given by PK_SURF_ask_uvbox. For a non-periodic parameter, the high value also lies in that range. For a periodic parameter, the high value will lie in the period above the low value if the required uvbox crosses the seam.

The only exception to the above occurs when a face lies partly on the extension of its surface, in which case the uvbox returned by PK_FACE_find_uvbox will contain parameters outside the base range of the surface.

PK_VECTOR1_t

```
typedef struct PK_VECTOR_s PK_VECTOR1_t;
```

This structure represents a unit 3 space vector.

Specific Errors

```
PK_ERROR_not_a_unit_vector      vector magnitude is not 1.0
```

This type is identical to PK_VECTOR_t except that the vector magnitude must be 1.0

PK_VECTOR_t

```
struct PK_VECTOR_s
    {
    double coord[3];
    };
typedef struct PK_VECTOR_s PK_VECTOR_t;
```

This structure represents a 3 space vector.

The components are as follows:

```
'coord[0]'  X component of vector
'coord[1]'  Y component of vector
'coord[2]'  Z component of vector
```

PK_VERTEX_array_t

This structure contains an array of entities of type PK_VERTEX_t

```
typedef struct PK_VERTEX_array_s
    {
    PK_VERTEX_t    *array;   --- array of VERTEXs
    int             length;  --- length of array
    }
    PK_VERTEX_array_t;
```

It has the fields:

'array': array of entities of type PK_VERTEX_t

'length': the number of entities in 'array'

PK_VERTEX_t

This type represents a VERTEX. A vertex is a single position in 3D space.

A vertex is a type of topology (PK_CLASS_vertex is a subclass of PK_CLASS_topol).

PK_VERTEX_type_t

typedef int PK_VERTEX_type_t;

PK_VERTEX_type_isolated_c	isolated vertex (has no edges)
PK_VERTEX_type_spur_c	spur vertex (single edge, one end only)
PK_VERTEX_type_wire_c	wire vertex (all edges are wires)
PK_VERTEX_type_normal_c	normal vertex

PK Tokens (Numeric) A

A.1 Numerical Listing of PK Interface Tokens

This appendix lists all the PK tokens numerically, and where appropriate cross-refers to the KI token.

By convention, PK tokens end in _c, indicating that the token is a constant. Those that don't end in _c (for instance the clash type tokens returned by PK_TOPOL_clash) don't conform to the convention.

Number	PK Token	KI Token
1	PK_abort_user_interrupt_c	SLABUI
2	PK_abort_runtime_error_c	SLABRE
3	PK_abort_frustrum_error_c	SLABFE

Number	PK Token
500	PK_CLASS_null
501	PK_CLASS_class
502	PK_CLASS_primitive
503	PK_CLASS_error
504	PK_CLASS_session
505	PK_CLASS_memory
506	PK_CLASS_mark
507	PK_CLASS_pmark
508	PK_CLASS_partition
509	PK_CLASS_bb
510	PK_CLASS_int
511	PK_CLASS_double
512	PK_CLASS_char
513	PK_CLASS_string

Number	PK Token
514	PK_CLASS_logical
515	PK_CLASS_vector
516	PK_CLASS_interval
517	PK_CLASS_box
518	PK_CLASS_uvbox
519	PK_CLASS_uv
520	PK_CLASS_pointer
521	PK_CLASS_vector1
522	PK_CLASS_size

Number	PK Token	KI Token
1000	PK_CLASS_entity	TYEN00
1001	PK_CLASS_geom	TYENGE
1002	PK_CLASS_topol	TYENTO
1003	PK_CLASS_attrib	TYENAD

Number	PK Token	KI Token
2002	PK_CLASS_curve	TYGECU
2003	PK_CLASS_surf	TYGESU
2004	PK_CLASS_transf	TYGETF
2005	PK_CLASS_nabox	
2501	PK_CLASS_point	TYPTCA

Number	PK Token	KI Token
3001	PK_CLASS_line	TYCUST
3002	PK_CLASS_circle	TYCUCI
3003	PK_CLASS_ellipse	TYCUEL
3004	PK_CLASS_icurve	TYCUIN
3005	PK_CLASS_bcurve	TYCUPA
3006	PK_CLASS_spcurve	TYCUSP
3007	PK_CLASS_fcurve	TYCUFG

Number	PK Token	KI Token
3008	PK_CLASS_cpcurve	TYCUCP
3009	PK_CLASS_trcurve	TYCUTR
3500	PK_SHELL_type_acorn_c	
3501	PK_SHELL_type_wireframe_c	
3502	PK_SHELL_type_wireframe_c	
3503	PK_SHELL_type_mixed_c	
3550	PK_SHELL_sign_positive_c	
3551	PK_SHELL_sign_negative_c	
3552	PK_SHELL_sign_open_c	
3600	PK_FIN_type_wire_c	
3601	PK_FIN_type_biwire_c	
3602	PK_FIN_type_normal_c	
3650	PK_EDGE_type_open_c	
3651	PK_EDGE_type_closed_c	
3652	PK_EDGE_type_ring_c	
3700	PK_EDGE_type_wireframe_c	
3701	PK_EDGE_type_laminar_c	
3702	PK_EDGE_type_normal_c	
3703	PK_EDGE_type_general_c	
3800	PK_neutral_success_c	
3801	PK_neutral_illegal_input_c	
3802	PK_neutral_numerical_failure_c	
3803	PK_neutral_unknown_config_c	
3804	PK_neutral_imprint_failure_c	

Number	PK Token	KI Token
4001	PK_CLASS_plane	TYSUPL
4002	PK_CLASS_cyl	TYSUCY
4003	PK_CLASS_cone	TYSUCO
4004	PK_CLASS_sphere	TYSUSP
4005	PK_CLASS_torus	TYSUTO
4006	PK_CLASS_bsurf	TYSUPA

Number	PK Token	KI Token
4007	PK_CLASS_blendsf	TYSUBL
4008	PK_CLASS_offset	TYSUOF
4009	PK_CLASS_swept	TYSUSE
4010	PK_CLASS_spun	TYSUSU
4011	PK_CLASS_fsurf	TYSUFG

Number	PK Token	KI Token
5001	PK_CLASS_vertex	TYTOVX
5002	PK_CLASS_edge	TYTOED
5003	PK_CLASS_loop	TYTOLO
5004	PK_CLASS_face	TYTOFA
5005	PK_CLASS_shell	TYTOSH
5006	PK_CLASS_body	TYTOBY
5007	PK_CLASS_instance	TYTOIN
5008	PK_CLASS_assembly	TYTOAS
5010	PK_CLASS_fin	TYTOFN
5011	PK_CLASS_region	TYTORG
5012	PK_CLASS_part	
5101	PK_VERTEX_type_isolated_c	ENVEIS
5102	PK_VERTEX_type_spur_c	ENVESP
5103	PK_VERTEX_type_wire_c	ENVEWR
5104	PK_VERTEX_type_normal_c	ENVENO
5401	PK_LOOP_type_hole_c	ENLOHO
5402	PK_LOOP_type_peripheral_c	ENLOPE
5403	PK_LOOP_type_other_c	ENLONA
5410	PK_LOOP_type_vertex_c	
5411	PK_LOOP_type_wire_c	
5412	PK_LOOP_type_outer_c	
5413	PK_LOOP_type_inner_c	
5414	PK_LOOP_type_winding_c	
5415	PK_LOOP_type_inner_sing_c	
5416	PK_LOOP_type_likely_outer_c	

Number	PK Token	KI Token
5417	PK_LOOP_type_likely_inner_c	
5418	PK_LOOP_type_unclear_c	
5419	PK_LOOP_type_error_c	
5601	PK_BODY_type_solid_c	ENBYSO
5602	PK_BODY_type_sheet_c	ENBYSH
5603	PK_BODY_type_minimum_c	ENBYMN
5604	PK_BODY_type_wire_c	ENBYWR
5605	PK_BODY_type_general_c	ENBYGN
5606	PK_BODY_type_acorn_c	
5701	PK_enclosure_inside_c	ENCLIN
5702	PK_enclosure_outside_c	ENCLOU
5703	PK_enclosure_on_c	ENCLON
5901	PK_ATTRIB_field_integer_c	RQAPIN
5902	PK_ATTRIB_field_real_c	RQAPRL
5903	PK_ATTRIB_field_string_c	RQAPCS
5094	PK_ATTRIB_field_vector_c	RQAPVC
5905	PK_ATTRIB_field_coordinate_c	RQAPCO
5906	PK_ATTRIB_field_direction_c	RQAPDR
5907	PK_ATTRIB_field_axis_c	RQAPAX
5908	PK_ATTRIB_field_entity_c	
5909	PK_ATTRIB_field_pointer_c	
5910	PK_ATTRIB_field_ustring_c	

Number	PK Token	KI Token
6002	PK_CLASS_KI_list	TYADLI
6003	PK_CLASS_attdef	TYADAD
6005	PK_CLASS_group	TYADFE

Number	PK Token
8500	PK_knot_unset_c
8501	PK_knot_non_uniform_c
8502	PK_knot_uniform_c

Number	PK Token
8503	PK_knot_quasi_uniform_c
8504	PK_knot_piecewise_bezier_c
8505	PK_knot_bezier_ends_c
8506	PK_knot_smooth_seam_c
8550	PK_self_intersect_unset_c
8551	PK_self_intersect_false_c
8552	PK_self_intersect_true_c
8600	PK_convexity_unset_c
8601	PK_convexity_arbitrary_c
8602	PK_convexity_convex_c
8603	PK_convexity_concave_c
8650	PK_BCURVE_form_unset_c
8651	PK_BCURVE_form_arbitrary_c
8652	PK_BCURVE_form_polyline_c
8653	PK_BCURVE_form_circular_c
8654	PK_BCURVE_form_elliptic_c
8655	PK_BCURVE_form_parabolic_c
8656	PK_BCURVE_form_hyperbolic_c
8700	PK_BSURF_form_unset_c
8701	PK_BSURF_form_arbitrary_c
8702	PK_BSURF_form_planar_c
8703	PK_BSURF_form_cylindrical_c
8704	PK_BSURF_form_conical_c
8705	PK_BSURF_form_spherical_c
8706	PK_BSURF_form_toroidal_c
8707	PK_BSURF_form_revolved_c
8708	PK_BSURF_form_ruled_c
8709	PK_BSURF_form_gen_cone_c
8710	PK_BSURF_form_quadric_c
8711	PK_BSURF_form_swept_c

Number	PK Token	KI Token
13801	PK_BODY_state_inside_out_c	RTSTNG
13802	PK_BODY_state_corrupt_c	RTSTCR
13803	PK_TOPOL_state_no_geom_c	RTSTMG
13804	PK_FACE_state_self_int_c	RTSTSX
13805	PK_GEOM_state_self_int_c	RTSTGX
13806	PK_GEOM_state_degenerate_c	RTSTDG
13808	PK_ENTITY_state_invalid_c	RTSTIG
13810	PK_TOPOL_state_size_box_c	RTSTBX
13812	PK_TOPOL_state_check_fail_c	RTSTCF
13816	PK_FACE_state_bad_face_face_c	RTSTFF
13817	PK_EDGE_state_open_c	RTSTOC
13818	PK_EDGE_state_bad_vertex_c	RTSTVC
13819	PK_EDGE_state_reversed_c	RTSTER
13820	PK_EDGE_state_bad_spcurve_c	RTSTSP
13821	PK_EDGE_state_vertices_touch_c	RTSTVT
13822	PK_FACE_state_bad_vertex_c	RTSTVS
13823	PK_FACE_state_bad_edge_c	RTSTES
13824	PK_EDGE_state_bad_edge_c	RTSTEO
13825	PK_FACE_state_no_vtx_at_sing_c	RTSTMV
13826	PK_FACE_state_bad_loops_c	RTSTLC
13827	PK_TOPOL_state_not_G1_c	RTSTGC
13828	PK_REGION_state_bad_shells_c	RTSTSH
13829	PK_FACE_state_check_fail_c	RTSTFC
13830	PK_FACE_state_bad_wire_fa_ed_c	RTSTEF
13831	PK_EDGE_state_bad_wire_ed_ed_c	RTSTEE
13832	PK_EDGE_state_bad_face_order_c	RTSTFO
13833	PK_SHELL_state_bad_topol_geom_c	RTSTSG
13834	PK_SHELL_state_bad_sh_sh_c	RTSTAC
13835	PK_BODY_state_bad_regions_c	RTSTRS
13836	PK_BODY_state_invalid_ident_c	RTSTID
13837	PK_EDGE_state_open_nmnl_c	RTSTON
13838	PK_EDGE_state_bad_vertex_nmnl_c	RTSTVN

Number	PK Token	KI Token
13839	PK_EDGE_state_bad_spcurve_nmnl_c	RTSTSN
13840	PK_EDGE_state_reversed_nmnl_c	RTSTRN
13841	PK_ATTDEF_state_bad_name_c	RTSTAN
13842	PK_ATTRIB_state_bad_string_c	RTSTAS
13903	PK_FACE_trim_surf_bsurf_c	SROPBS
13905	PK_FACE_trim_confine_no_c	SROPCN
13906	PK_FACE_trim_confine_yes_c	SROPCY
13907	PK_FACE_trim_confine_closed_c	SROPCC
13908	PK_FACE_trim_confine_periph_c	SROPCP
13909	PK_FACE_trim_degen_yes_c	SROPID
13910	PK_FACE_trim_degen_no_c	SROPED

Number	PK Token	KI Token
14001	PK_mass_no_c	MAOPNA
14002	PK_mass_mass_c	MAOPAM
14003	PK_mass_c_of_g_c	MAOPCG
14004	PK_mass_m_of_i_c	MAOPIN
14005	PK_mass_periphery_no_c	MAOPNP
14006	PK_mass_periphery_yes_c	MAOPPE
14007	PK_mass_bound_no_c	MAOPNE
14008	PK_mass_bound_modulus_c	MAOPEM
14009	PK_mass_bound_interval_c	MAOPEI
14401	PK_BB_event_create_c	BBEVCR
14402	PK_BB_event_delete_c	BBEVDE
14403	PK_BB_event_change_c	BBEVCH
14404	PK_BB_event_split_c	BBEVSP
14405	PK_BB_event_merge_c	BBEVME
14406	PK_BB_event_transfer_c	BBEVTR
14407	PK_BB_event_copy_c	BBEVCO
14408	PK_BB_event_transform_c	BBEVTF
14409	PK_BB_event_change_attrib_c	BBEVAC
14501	PK_BB_status_off_c	BBOPOF

Number	PK Token	KI Token
14502	PK_BB_status_on_c	BBOPON
14503	PK_BB_status_user_field_c	BBOPUF
14611	PK_intersect_vector_simple_c	CICLSI
14612	PK_intersect_vector_tangent_c	CICLTG
14613	PK_intersect_vector_start_c	CICLSC
14614	PK_intersect_vector_end_c	CICLEC
14651	PK_intersect_curve_simple_c	SICLSI
14652	PK_intersect_curve_tangent_c	SICLTG
14801	PK_intersect_fc_simple_c	CFCLSI
14802	PK_intersect_fc_tangent_c	CFCLTG
14803	PK_intersect_fc_out_in_c	CFCLEF
14804	PK_intersect_fc_in_out_c	CFCLLF
14805	PK_intersect_fc_out_coi_c	CFCLEB
14806	PK_intersect_fc_coi_out_c	CFCLLB
14807	PK_intersect_fc_coi_in_c	CFCLEI
14808	PK_intersect_fc_in_coi_c	CFCLLI
14809	PK_intersect_fc_in_tangent_c	CFCLTI
14810	PK_intersect_fc_out_tangent_c	CFCLTO
14811	PK_intersect_fc_in_c	CFCLUC
14812	PK_intersect_fc_start_c	CFCLSC
14813	PK_intersect_fc_end_c	CFCLEC

Number	PK Token	KI Token
15101	PK_BODY_fault_no_fault_c	RTTOOK
15102	PK_BODY_fault_bad_body_c	RTTOBB
15103	PK_BODY_fault_unknown_c	RTTODE
15104	PK_BODY_fault_unknown_child_c	RTTOUC
15105	PK_BODY_fault_duplicate_child_c	RTTODC
15106	PK_BODY_fault_wrong_child_c	RTTOWC
15107	PK_BODY_fault_missing_child_c	RTTOFC
15108	PK_BODY_fault_extra_child_c	RTTOMC
15109	PK_BODY_fault_wrong_parent_c	RTTOWP

Number	PK Token	KI Token
15110	PK_BODY_fault_missing_parent_c	RTTOFP
15111	PK_BODY_fault_extra_parent_c	RTTOMP
15112	PK_BODY_fault_bad_wire_c	RTTODW
15113	PK_BODY_fault_bad_loop_c	RTTOIL
15114	PK_BODY_fault_bad_shells_c	RTTOCS
15115	PK_BODY_fault_disjoint_shell_c	RTTODS
15116	PK_BODY_fault_non_manifold_c	RTTONM
15701	PK_BODY_knit_completed_c	RTKNOK
15702	PK_BODY_knit_not_completed_c	RTKNIN
15901	PK_boolean_intersect_c	BOOPIN
15902	PK_boolean_subtract_c	BOOPSU
15903	PK_boolean_unite_c	BOOPUN
15905	PK_boolean_exclude_c	BOOPEC
15906	PK_boolean_include_c	BOOPIC
15949	PK_boolean_off_c	

Number	PK Token	KI Token
16051	PK_blend_fault_singularity_c	BLCCSN
16052	PK_blend_fault_obsolete_c	BLCCOT
16053	PK_blend_fault_vertex_c	BLCCMX
16054	PK_blend_fault_sheet_c	BLCCRS
16055	PK_blend_fault_general_c	BLCCRE
16056	PK_blend_fault_2_edge_c	BLCCTV
16057	PK_blend_fault_chamfer_c	BLCCHM
16058	PK_blend_fault_bsurf_c	BLCCXT
16059	PK_blend_fault_range_c	BLCCIR
16060	PK_blend_fault_type_c	BLCCIT
16061	PK_blend_fault_edge_c	BLCCAB
16062	PK_blend_fault_loop_c	BLCCOL
16063	PK_blend_fault_overlap_c	BLCCOB
16064	PK_blend_fault_overlap_edge_c	BLCCOU
16065	PK_blend_fault_unknown_c	BLCCUN

Number	PK Token	KI Token
16066	PK_blend_fault_bad_end_c	BLCCUE
16067	PK_blend_fault_face_c	BLCCRL
16068	PK_blend_fault_overlap_end_c	BLCCOE
16069	PK_blend_fault_end_c	BLCCIE
16070	PK_blend_fault_chamfer_intsec_c	BLCCIX
16071	PK_blend_fault_edge_intsec_c	BLCCEX
16072	PK_blend_fault_other_edge_c	BLCCOI
16073	PK_blend_fault_tangent_c	BLCCTN
16074	PK_blend_fault_cliffedge_c	BLCCIP
16401	PK_BODY_state_ok_c	RTTSOK
16402	PK_FACE_state_redundant_c	RTTSFR
16403	PK_CURVE_state_inconsistent_c	RTTSCI
16405	PK_LOOP_state_invalid_c	RTTSLI
16406	PK_EDGE_state_bad_order_c	RTTSEO

Number	PK Token	KI Token
17201	PK_reset_prec_ok_c	RTTLOK
17202	PK_reset_prec_tangent_c	RTTLNT
17203	PK_reset_prec_missing_geom_c	RTTLMG
17204	PK_reset_prec_failure_c	RTTLRF
17401	PK_blend_trim_no_c	FXFTNT
17402	PK_blend_trim_to_walls_c	FXFTTB
17403	PK_blend_walls_trim_both_c	FXFTTW
17404	PK_blend_walls_attach_c	FXFTAT
17417	PK_blend_walls_solid_c	FXFTSO
17420	PK_blend_trim_short_c	FXFTST
17421	PK_blend_trim_long_c	FXFTLT
17451	PK_fxf_fault_no_fault_c	FXFEOK
17452	PK_fxf_fault_sheet_c	FXFEST
17453	PK_fxf_fault_unknown_c	FXFEER
17454	PK_fxf_fault_insufficient_c	FXFEID
17455	PK_fxf_fault_inconsistent_c	FXFEXD

Number	PK Token	KI Token
17456	PK_fxf_fault_wall_c	FXFEIF
17457	PK_fxf_fault_range_c	FXFEIR
17458	PK_fxf_fault_thl_c	FXFEIH
17459	PK_fxf_fault_cliff_c	FXFEIC
17460	PK_fxf_fault_curved_c	FXFEFC
17461	PK_fxf_fault_small_c	FXFERS
17462	PK_fxf_fault_large_c	FXFERL
17463	PK_fxf_fault_left_c	FXFELN
17464	PK_fxf_fault_right_c	FXFERN
17465	PK_fxf_fault_both_c	FXFEBN
17466	PK_fxf_fault_sheet_clash_c	FXFESC
17467	PK_fxf_fault_wall_clash_c	FXFEWC
17468	PK_fxf_fault_self_int_c	FXFEGX
17469	PK_fxf_fault_face_face_c	FXFEFF
17470	PK_fxf_fault_rho_value_c	FXFERV
17471	PK_fxf_fault_asymmetric_c	FXFEAR
17472	PK_fxf_fault_chl_c	FXFECL
17473	PK_fxf_fault_bad_spine_c	FXFEIS
17474	PK_fxf_fault_bad_ribs_c	

Number	PK Token
18000	PK_PARAM_bound_infinite_c
18001	PK_PARAM_bound_extendable_c
18002	PK_PARAM_bound_bound_c
18003	PK_PARAM_bound_closed_c
18004	PK_PARAM_bound_degenerate_c
18020	PK_PARAM_periodic_no_c
18021	PK_PARAM_periodic_yes_c
18022	PK_PARAM_periodic_seamed_c
18040	PK_PARAM_form_linear_c
18041	PK_PARAM_form_circular_c
18042	PK_PARAM_form_any_c

Number	PK Token
18060	PK_BODY_sewing_overlapping_c
18061	PK_BODY_sewing_non_manifold_c
18062	PK_BODY_sewing_non_oriented_c
18063	PK_BODY_sewing_unspecified_c
18070	PK_BODY_sewing_solid_c
18071	PK_BODY_sewing_sheet_c
18072	PK_BODY_sewing_general_c
18073	PK_BODY_sewing_any_c
18076	PK_BODY_sewing_remove_none_c
18077	PK_BODY_sewing_remove_poss_c
18078	PK_BODY_sewing_remove_cert_c
18080	PK_FACE_heal_no_c
18081	PK_FACE_heal_cap_c
18082	PK_FACE_heal_grow_from_child_c
18083	PK_FACE_heal_grow_from_parent_c
18084	PK_FACE_heal_shrink_c
18085	PK_FACE_heal_yes_c
18100	PK_local_check_no_c
18101	PK_local_check_ok_c
18102	PK_local_check_negated_c
18103	PK_local_check_failed_c
18210	PK_boolean_fence_front_c
18211	PK_boolean_fence_back_c
18212	PK_boolean_fence_none_c
18200	PK_section_fence_front_c
18201	PK_section_fence_back_c
18202	PK_section_fence_both_c
18220	PK_transmit_format_text_c
18221	PK_transmit_format_binary_c
18222	PK_transmit_format_neutral_c
18223	PK_transmit_format_applio_c
18224	PK_transmit_format_xml_c

Number	PK Token
18230	PK_PARTITION_xmt_deltas_none_c
18231	PK_PARTITION_xmt_deltas_all_c
18232	PK_PARTITION_xmt_deltas_main_c
18240	PK_PARTITION_rcv_deltas_no_c
18241	PK_PARTITION_rcv_deltas_later_c
18250	PK_boolean_match_exact_c
18251	PK_boolean_match_contains_c
18252	PK_boolean_match_overlap_c
18253	PK_boolean_match_imprinted_c
18260	PK_range_guess_no_c
18261	PK_range_guess_param_c
18262	PK_range_guess_vector_c
18270	PK_range_result_found_c
18271	PK_range_result_lower_c
18272	PK_range_result_upper_c
18280	PK_check_geom_no_c
18281	PK_check_geom_basic_c
18282	PK_check_geom_lazy_c
18283	PK_check_geom_full_c
18284	PK_check_geom_yes_c
18290	PK_check_bgeom_no_c
18291	PK_check_bgeom_yes_c
18300	PK_check_top_geo_no_c
18301	PK_check_top_geo_edge_c
18302	PK_check_top_geo_face_c
18303	PK_check_top_geo_yes_c
18310	PK_check_size_box_no_c
18311	PK_check_size_box_yes_c
18320	PK_check_fa_X_no_c
18321	PK_check_fa_X_yes_c
18330	PK_check_loops_no_c
18331	PK_check_loops_yes_c

Number	PK Token
18340	PK_check_fa_fa_no_c
18341	PK_check_fa_fa_yes_c
18350	PK_check_sh_no_c
18351	PK_check_sh_yes_c
18360	PK_check_corrupt_no_c
18361	PK_check_corrupt_yes_c
18370	PK_FACE_trim_surf_own_c
18380	PK_blend_check_vertex_c
18381	PK_blend_check_full_c
18390	PK_blend_fault_no_blend_c
18391	PK_blend_fault_no_fault_c
18392	PK_blend_fault_self_int_c
18393	PK_blend_fault_face_face_c
18394	PK_blend_fault_conic_c
18400	PK_blend_type_no_c
18401	PK_blend_type_ball_constant_c
18402	PK_blend_type_ball_variable_c
18403	PK_blend_type_chamfer_c
18410	PK_blend_propagate_c
18411	PK_blend_propagate_yes_c
18420	PK_blend_vary_smooth_c
18421	PK_blend_vary_linear_c
18430	PK_blend_render_rib_no_c
18431	PK_blend_render_rib_yes_c
18440	PK_blend_ov_smooth_no_c
18441	PK_blend_ov_smooth_diff_c
18442	PK_blend_ov_smooth_any_c
18450	PK_blend_ov_cliff_no_c
18451	PK_blend_ov_cliff_diff_c
18452	PK_blend_ov_cliff_any_c
18460	PK_blend_ov_cliff_end_no_c
18461	PK_blend_ov_cliff_end_yes_c

Number	PK Token
18470	PK_blend_ov_notch_no_c
18471	PK_blend_ov_notch_yes_c
18480	PK_blend_walls_trim_no_c
18490	PK_blend_xs_rolling_ball_c
18491	PK_blend_xs_disc_c
18492	PK_blend_xs_isoparameter_c
18500	PK_piecewise_rep_bezier_c
18501	PK_piecewise_rep_polynomial_c
18502	PK_piecewise_rep_hermite_c
18503	PK_piecewise_rep_taylor_c
18510	PK_PARAM_end_periodic_c
18511	PK_PARAM_end_natural_c
18512	PK_PARAM_end_clamped_vector_c
18513	PK_PARAM_end_bcurve_c
18520	PK_PARAM_knot_auto_c
18521	PK_PARAM_knot_defined_c
18530	PK_PARAM_twist_no_c
18531	PK_PARAM_twist_yes_c
18540	PK_TOPOL_sense_none_c
18541	PK_TOPOL_sense_negative_c
18542	PK_TOPOL_sense_positive_c
18550	PK_PARAM_degen_no_c
18551	PK_PARAM_degen_vector_c
18552	PK_PARAM_degen_bcurve_c

Number	PK Token
20100	PK_render_edge_no_c
20101	PK_render_edge_yes_c
20120	PK_render_silhouette_no_c
20121	PK_render_silhouette_arcs_c
20122	PK_render_silhouette_yes_c
20140	PK_render_planar_no_c

Number	PK Token
20141	PK_render_planar_attrib_c
20142	PK_render_planar_free_c
20143	PK_render_planar_yes_c
20160	PK_render_radial_no_c
20161	PK_render_radial_attrib_c
20162	PK_render_radial_yes_c
20163	PK_render_radial_free_c
20180	PK_render_param_no_c
20181	PK_render_param_attrib_c
20182	PK_render_param_yes_c
20183	PK_render_param_free_c
20200	PK_render_unfix_no_c
20201	PK_render_unfix_attrib_c
20202	PK_render_unfix_yes_c
20220	PK_render_boundary_no_c
20221	PK_render_boundary_yes_c
20300	PK_render_vis_no_c
20301	PK_render_vis_hid_c
20302	PK_render_vis_inv_c
20303	PK_render_vis_hid_draft_c
20304	PK_render_vis_inv_draft_c
20305	PK_render_vis_extended_c
20320	PK_render_smooth_no_c
20321	PK_render_smooth_yes_c
20322	PK_render_smooth_draft_c
20340	PK_render_internal_no_c
20341	PK_render_internal_yes_c
20360	PK_render_region_no_c
20361	PK_render_region_attrib_c
20362	PK_render_region_yes_c
20380	PK_render_hierarch_no_c
20381	PK_render_hierarch_no_geom_c

Number	PK Token
20382	PK_render_hierarch_yes_c
20383	PK_render_hierarch_param_c
20400	PK_render_bcurve_polyline_c
20401	PK_render_bcurve_bezier_c
20402	PK_render_bcurve_nurbs_c
20420	PK_render_viewport_no_c
20421	PK_render_viewport_yes_c
20422	PK_render_viewport_array_c
20440	PK_render_invisible_no_c
20441	PK_render_invisible_yes_c
20450	PK_render_drafting_no_c
20451	PK_render_drafting_yes_c
20460	PK_render_self_hidden_no_c
20461	PK_render_self_hidden_yes_c
20500	PK_facet_shape_any_c
20501	PK_facet_shape_cut_c
20502	PK_facet_shape_convex_c
20520	PK_facet_match_no_c
20521	PK_facet_match_geom_c
20522	PK_facet_match_topol_c
20523	PK_facet_match_trimmed_c
20540	PK_facet_density_no_view_c
20541	PK_facet_density_use_view_c
20560	PK_facet_cull_no_c
20561	PK_facet_cull_back_c
20580	PK_facet_degen_multiple_vxs_c
20581	PK_facet_degen_single_vx_c
20582	PK_facet_degen_average_parms_c
20600	PK_facet_go_normals_no_c
20601	PK_facet_go_normals_yes_c
20620	PK_facet_go_parameters_no_c
20621	PK_facet_go_parameters_d0_c

Number	PK Token
20622	PK_facet_go_parameters_d1_c
20623	PK_facet_go_parameters_d2_c
20640	PK_facet_go_edges_no_c
20641	PK_facet_go_edges_yes_c
20660	PK_facet_go_strips_no_c
20661	PK_facet_go_strips_yes_c
20680	PK_facet_go_interleaved_no_c
20681	PK_facet_go_interleaved_yes_c
20700	PK_facet_fault_face_c
20701	PK_facet_fault_facet_c
20702	PK_facet_fault_vertex_point_c
20703	PK_facet_fault_vertex_normal_c
20704	PK_facet_fault_vertex_param_c
20705	PK_facet_fault_thin_face_c
20706	PK_facet_fault_body_c
20900	PK_BODY_pick_radial_c
20901	PK_BODY_pick_axial_c
20902	PK_BODY_pick_ratio_c

Number	PK Token
21000	PK_boolean_select_specific_c
21001	PK_boolean_select_adjacent_c
21002	PK_boolean_select_propagate_c
21010	PK_thread_disabled_c
21011	PK_thread_per_processor_c
21012	PK_thread_absolute_c
21020	PK_blend_output_rib_no_c
21021	PK_blend_output_rib_on_fail_c
21022	PK_blend_output_rib_only_c
21023	PK_blend_output_rib_also_c
21030	PK_blend_group_rib_no_c
21031	PK_blend_group_rib_by_face_c

Number	PK Token
21032	PK_blend_group_rib_by_parms_c
21040	PK_blend_check_su_X_no_c
21041	PK_blend_check_su_X_not_bsurf_c
21042	PK_blend_check_su_X_yes_c
21050	PK_blend_check_fa_no_c
21051	PK_blend_check_fa_yes_c
21060	PK_blend_check_fa_fa_no_c
21061	PK_blend_check_fa_fa_yes_c
21070	PK_blend_update_0_c
21071	PK_blend_update_1_c
21080	PK_bcurve_fit_eval_user_c
21081	PK_bcurve_fit_eval_chain_c
21090	PK_bcurve_fit_err_none_c
21091	PK_bcurve_fit_err_parm_c
21092	PK_bcurve_fit_err_closest_c
21100	PK_bcurve_fit_success_c
21101	PK_bcurve_fit_failure_c
21102	PK_bcurve_fit_other_failed_c
21103	PK_bcurve_fit_eval_failure_c
21104	PK_bcurve_fit_bad_n_curves_c
21105	PK_bcurve_fit_bad_range_c
21106	PK_bcurve_fit_bad_tolerance_c
21107	PK_bcurve_fit_bad_err_method_c
21108	PK_bcurve_fit_bad_eval_type_c
21109	PK_bcurve_fit_bad_break_parm_c
21110	PK_bcurve_fit_bad_chain_c
21111	PK_bcurve_fit_bad_fit_parm_c
21200	PK_BODY_curve_nmnl_no_c
21201	PK_BODY_curve_nmnl_yes_c
21220	PK_BODY_owb_gap_fill_round_c
21221	PK_BODY_owb_gap_fill_linear_c
21222	PK_BODY_owb_gap_fill_natural_c

Number	PK Token
21300	PK_pattern_check_fa_fa_no_c
21301	PK_pattern_check_fa_fa_yes_c
21310	PK_pattern_check_loops_no_c
21311	PK_pattern_check_loops_yes_c
21320	PK_pattern_status_ok_c
21321	PK_pattern_status_fail_c
21322	PK_pattern_status_outside_fa_c
21323	PK_pattern_status_outside_reg_c
21324	PK_pattern_status_tf_failed_c
21325	PK_pattern_status_face_clash_c
21360	PK_pattern_result_ok_c
21361	PK_pattern_result_partial_c
21362	PK_pattern_result_fail_c
21363	PK_pattern_result_mixed_faces_c
21364	PK_pattern_result_instanced_c
21365	PK_pattern_result_invalid_set_c
21366	PK_pattern_result_miss_geom_c
21400	PK_local_check_su_X_no_c
21401	PK_local_check_su_X_yes_c
21410	PK_local_check_topol_no_c
21411	PK_local_check_topol_yes_c
21420	PK_local_change_topol_no_c
21421	PK_local_change_topol_yes_c
21450	PK_local_status_ok_c
21451	PK_local_status_nocheck_c
21452	PK_local_status_fail_c
21453	PK_local_status_cant_get_pt_c
21454	PK_local_status_cant_get_cu_c
21455	PK_local_status_cant_get_su_c
21456	PK_local_status_cant_offset_c
21457	PK_local_status_side_cu_fail_c
21458	PK_local_status_side_su_fail_c

Number	PK Token
21459	PK_local_status_not_same_by_c
21460	PK_local_status_fa_fail_c
21461	PK_local_status_fa_fa_fail_c
21462	PK_local_status_ed_remains_c
21463	PK_local_status_point_contact_c
21464	PK_local_status_bad_reference_c
21465	PK_local_status_not_supported_c
21500	PK_TOPOL_track_delete_c
21501	PK_TOPOL_track_merge_c
21502	PK_TOPOL_track_create_c
21503	PK_TOPOL_track_split_c
21504	PK_TOPOL_track_derive_c
21520	PK_BODY_simplify_yes_c
21521	PK_BODY_simplify_no_c
21530	PK_BODY_curv_natural_c
21531	PK_BODY_curv_unconstrained_no_c
21540	PK_BODY_loft_no_clamp_c
21541	PK_BODY_loft_vector_clamp_c
21542	PK_BODY_loft_face_clamp_c
21543	PK_BODY_loft_clamp_planar_c
21550	PK_BODY_loft_ok_c
21551	PK_BODY_loft_failure_c
21552	PK_BODY_loft_bad_profile_c
21553	PK_BODY_loft_surf_failure_c
21554	PK_BODY_loft_check_failure_c
21555	PK_BODY_loft_geom_self_int_c
21556	PK_BODY_loft_topol_self_int_c
21557	PK_BODY_loft_geom_degen_c
21558	PK_BODY_loft_small_profile_c
21560	PK_BODY_sweep_align_normal_c
21561	PK_BODY_sweep_align_parallel_c
21570	PK_BODY_sweep_law_none_c

Number	PK Token
21571	PK_BODY_sweep_law_discrete_c
21572	PK_BODY_sweep_law_curve_c
21580	PK_BODY_sweep_ok_c
21581	PK_BODY_sweep_failure_c
21582	PK_BODY_sweep_bad_path_c
21583	PK_BODY_sweep_bad_profile_c
21584	PK_BODY_sweep_surf_failure_c
21585	PK_BODY_sweep_curve_failure_c
21586	PK_BODY_sweep_check_failure_c
21587	PK_BODY_sweep_geom_self_int_c
21588	PK_BODY_sweep_topol_self_int_c
21589	PK_BODY_sweep_geom_degen_c
21590	PK_BODY_sweep_small_profile_c
21600	PK_boolean_report_failure_c
21601	PK_boolean_report_opposed_sheets_c
21602	PK_boolean_report_solid_has_void_c
21603	PK_boolean_report_partial_coi_c
21604	PK_boolean_report_t_sheet_c
21605	PK_boolean_report_non_manifold_c
21606	PK_boolean_report_invalid_match_c
21607	PK_boolean_report_imprint_fail_c
21608	PK_boolean_report_intersect_fail_c
21609	PK_boolean_report_Invalid_face_c
21650	PK_boolean_result_success_c
21651	PK_boolean_result_no_clash_c
21652	PK_boolean_result_no_effect_c
21653	PK_boolean_result_failed_c
21700	PK_section_report_failure_c
21701	PK_section_report_solid_has_void_c
21702	PK_section_report_partial_coi_c
21703	PK_section_report_t_sheet_c
21704	PK_section_report_non_manifold_c

Number	PK Token
21705	PK_section_report_invalid_match_c
21706	PK_section_report_imprint_fail_c
21707	PK_section_report_intersect_fail_c
21708	PK_section_report_invalid_face_c
21750	PK_section_result_success_c
21751	PK_section_result_no_clash_c
21752	PK_section_result_no_effect_c
21753	PK_section_result_failed_c
21770	PK_blend_transfer_topol_no_c
21771	PK_blend_transfer_topol_yes_c
21780	PK_local_check_fa_fa_no_c
21781	PK_local_check_fa_fa_yes_c
21800	PK_boolean_check_fa_no_c
21801	PK_boolean_check_fa_yes_c
21810	PK_section_check_fa_no_c
21811	PK_section_check_fa_yes_c
21820	PK_render_ignore_no_c
21821	PK_render_ignore_absolute_c
21822	PK_render_ignore_ratio_c
21823	PK_render_ignore_body_ratio_c
21830	PK_BODY_topology_minimal_c
21831	PK_BODY_topology_columns_c
21832	PK_BODY_topology_grid_c
21840	PK_step_tapered_c
21841	PK_step_normal_c
21845	PK_taper_method_curve_c
21846	PK_taper_method_surface_c
21847	PK_taper_method_isocline_c
21848	PK_taper_method_offset_c
21850	PK_bound_distance_c
21851	PK_bound_sheet_c
21852	PK_bound_surf_c

Number	PK Token
21853	PK_bound_face_c
21854	PK_bound_body_c
21871	PK_render_overlap_no_c
21872	PK_render_overlap_yes_c
21880	PK_FACE_coi_yes_c
21881	PK_FACE_coi_yes_reversed_c
21882	PK_FACE_coi_coi_no_topol_c
21883	PK_FACE_coi_coi_no_bound_1_c
21884	PK_FACE_coi_no_bound_2_c
21885	PK_FACE_coi_no_face_1_c
21886	PK_FACE_coi_no_face_2_c
21887	PK_FACE_coi_no_rubber_c
21900	PK_DEBUG_diff_type_all_c
21901	PK_DEBUG_diff_type_nums_c
21902	PK_DEBUG_diff_type_devs_c
21903	PK_DEBUG_diff_type_class_c
21904	PK_DEBUG_diff_type_tols_c
21905	PK_DEBUG_diff_type_sense_c
21906	PK_DEBUG_diff_type_miss_c
21907	PK_DEBUG_diff_type_non_match_c
2192	0PK_DEBUG_global_res_diffs_c
21921	PK_DEBUG_global_res_no_diffs_c
21930	PK_DEBUG_local_res_diffs_c
21931	PK_DEBUG_local_res_no_diffs_c
21940	PK_DEBUG_diff_n_shells_c
21941	PK_DEBUG_diff_n_faces_c
21942	PK_DEBUG_diff_n_loops_c
21943	PK_DEBUG_diff_n_acc_edges_c
21944	PK_DEBUG_diff_n_tol_edges_c
21945	PK_DEBUG_diff_n_fins_c
21946	PK_DEBUG_diff_n_acc_vxs_c
21947	PK_DEBUG_diff_n_tol_vxs_c

Number	PK Token
21948	PK_DEBUG_diff_n_vxs_c
21949	PK_DEBUG_diff_n_cht_pts_c
21950	PK_DEBUG_diff_surf_dev_c
21951	PK_DEBUG_diff_curve_dev_c
21952	PK_DEBUG_diff_vx_dev_c
21953	PK_DEBUG_diff_surf_class_c
21954	PK_DEBUG_diff_curve_class_c
21955	PK_DEBUG_diff_edge_tol_c
21956	PK_DEBUG_diff_vx_tol_c
21957	PK_DEBUG_diff_face_sense_c
21958	PK_DEBUG_diff_surf_sense_c
21959	PK_DEBUG_diff_curve_sense_c
21960	PK_DEBUG_diff_vx_missing_c
21961	PK_DEBUG_diff_face_match_c
21962	PK_DEBUG_diff_fin_match_c
21963	PK_DEBUG_diff_vx_match_c
21964	PK_DEBUG_local_surf_dev_c
21965	PK_DEBUG_local_curve_dev_c
21966	PK_DEBUG_local_vx_dev_c
21967	PK_DEBUG_local_surf_class_c
21968	PK_DEBUG_local_curve_class_c
21969	PK_DEBUG_local_edge_tol_c
21970	PK_DEBUG_local_vx_tol_c
21971	PK_DEBUG_local_face_sense_c
21972	PK_DEBUG_local_surf_sense_c
21973	PK_DEBUG_local_curve_sense_c
21974	PK_DEBUG_local_vx_missing_c
21975	PK_DEBUG_local_face_match_c
21976	PK_DEBUG_local_he_match_c
21977	PK_DEBUG_local_vx_match_c
21990	PK_boolean_match_style_basic_c
21991	PK_boolean_match_style_relax_c

Number	PK Token
22010	PK_SURF_extend_ok_c
22011	PK_SURF_extend_unextended_c
22012	PK_SURF_extend_partial_c
22013	PK_SURF_extend_invalid_c
22014	PK_SURF_extend_failure_c
22030	PK_SURF_extension_none_c
22031	PK_SURF_extension_point_c
22032	PK_SURF_extension_box_c
22033	PK_SURF_extension_uvbox_c
22034	PK_SURF_extension_ratio_c
22040	PK_thicken_method_offset_c
22041	PK_thicken_method_punch_c
22050	PK_ATTDEF_callback_normal_c
22051	PK_ATTDEF_callback_read_only_c
22060	PK_FACE_change_type_none_c
22061	PK_FACE_change_type_offset_c
22062	PK_FACE_change_type_taper_c
22063	PK_FACE_change_type_transform_c
22064	PK_FACE_change_type_replace_c
22065	PK_FACE_change_type_blend_c
22070	PK_PARAM_direction_u_c
22071	PK_PARAM_direction_v_c
22080	PK_fill_hole_ok_c
22081	PK_fill_hole_bad_edge_c
22082	PK_fill_hole_duplicate_c
22083	PK_fill_hole_vertex_c
22084	PK_fill_hole_too_many_loops_c
22085	PK_fill_hole_gap_c
22086	PK_fill_hole_no_edge_on_target_c
22087	PK_fill_hole_cant_match_body_c
22088	PK_fill_hole_cant_match_edge_c
22089	PK_fill_hole_too_small_c

Number	PK Token
22090	PK_fill_hole_unknown_c
22091	PK_fill_hole_face_c
22092	PK_fill_hole_face_face_c
22093	PK_fill_hole_not_smooth_c
22094	PK_fill_hole_too_complex_c
22100	PK_fill_hole_trim_to_hole_c
22101	PK_fill_hole_trim_to_sheet_c
22102	PK_fill_hole_create_patch_c
22111	PK_facet_ignore_no_c
22112	PK_facet_ignore_absolute_c
22113	PK_facet_ignore_ratio_c
22114	PK_facet_ignore_body_ratio_c
22120	PK_CURVE_approx_arclength_c
22129	PK_CURVE_approx_even_c
22130	PK_facet_ignore_scope_local_c
22131	PK_facet_ignore_scope_global_c
22140	PK_facet_wire_edges_no_c
22141	PK_facet_wire_edges_yes_c
22150	PK_offset_method_accurate_c
22151	PK_offset_method_approximate_c
22160	PK_render_memory_target_no_c
22161	PK_render_memory_target_yes_c
22170	PK_outline_project_no_c
22171	PK_outline_project_plane_c
22180	PK_pick_approximate_no_c
22181	PK_pick_approximate_yes_c
22190	PK_blend_edge_update_0_c
22191	PK_blend_edge_update_1_c
22200	PK_blend_xs_shape_unset_c
22201	PK_blend_xs_shape_conic_c
22202	PK_blend_xs_shape_g2_c
22203	PK_blend_xs_shape_chamfer_c

Number	PK Token
22210	PK_BODY_sweep_fair_no_c
22211	PK_BODY_sweep_fair_twist_c
22220	PK_comparison_always_c
22221	PK_comparison_less_c
22222	PK_comparison_equal_c
22223	PK_comparison_greater_c
22224	PK_comparison_between_c
22225	PK_comparison_outside_c
22226	PK_comparison_never_c
22240	PK_blend_identify_within_c
22241	PK_blend_identify_exc_chain_c
22242	PK_blend_identify_inc_chain_c
22243	PK_blend_identify_max_chain_c
22244	PK_blend_identify_dependent_c
22250	PK_detail_any_c
22251	PK_detail_rubber_c
22252	PK_detail_hole_cyl_c
22253	PK_detail_hole_cyl_through_c
22254	PK_detail_hole_cyl_blind_c
22255	PK_detail_hole_cyl_closed_c
22256	PK_detail_blend_rb_const_r_c
22300	PK_replace_merge_no_c
22301	PK_replace_merge_in_c
22302	PK_replace_merge_out_c
22310	PK_offset_step_no_c
22311	PK_offset_step_yes_c

Number	PK Token
32760	PK_HAND_left_c
32761	PK_HAND_right_c

PK Tokens (Alphabetic) B

B.1 Alphabetical List of PK Interface Tokens

This appendix contains all the PK tokens, listed alphabetically by class.

By convention, PK tokens end in _c, indicating that the token is a constant. Those that don't end in _c (for instance the clash type tokens returned by PK_TOPOL_clash) don't conform to the convention.

PK_abort_reason_t

PK_abort_user_interrupt_c	SLABUI
PK_abort_runtime_error_c	SLABRE
PK_abort_frustrum_error_c	SLABFE

PK_ATTDEF_class_t (RQAC00)

PK_ATTDEF_class_01_c	RQAC01
PK_ATTDEF_class_02_c	RQAC02
PK_ATTDEF_class_03_c	RQAC03
PK_ATTDEF_class_04_c	RQAC04
PK_ATTDEF_class_05_c	RQAC05
PK_ATTDEF_class_06_c	RQAC06
PK_ATTDEF_class_07_c	RQAC07

PK_ATTRIB_field_t (RQAP00)

PK_ATTRIB_field_real_c	RQAPRL
PK_ATTRIB_field_integer_c	RQAPIN
PK_ATTRIB_field_string_c	RQAPCS
PK_ATTRIB_field_vector_c	RQAPVC
PK_ATTRIB_field_coordinate_c	RQAPCO

PK_ATTRIB_field_direction_c	RQAPDR
PK_ATTRIB_field_axis_c	RQAPAX
PK_ATTRIB_field_entity_c	5908
PK_ATTRIB_field_pointer_c	5909
PK_ATTRIB_field_ustring_c	5910

PK_ATTDEF_callback_type_t

PK_ATTDEF_callback_normal_c	22050
PK_ATTDEF_callback_read_only_c	22051

PK_BB_status_t

PK_BB_status_off_c	BBOPOF
PK_BB_status_on_c	BBOPON
PK_BB_status_user_field_c	BBOPUF

PK_BB_event_t

PK_BB_event_create_c	BBEVCR
PK_BB_event_delete_c	BBEVDE
PK_BB_event_copy_c	BBEVCO
PK_BB_event_transfer_c	BBEVTR
PK_BB_event_merge_c	BBEVME
PK_BB_event_split_c	BBEVSP
PK_BB_event_transform_c	BBEVTF
PK_BB_event_change_attrib_c	BBEVAC
PK_BB_event_change_c	BBEVCH

PK_BCURVE_fit_t

PK_BCURVE_fit_success_c	21100
PK_BCURVE_fit_failure_c	21101
PK_BCURVE_fit_other_failed_c	21102
PK_BCURVE_fit_eval_failure_c	21103
PK_BCURVE_fit_bad_n_curves_c	21104

PK_BCURVE_fit_bad_range_c	21105
PK_BCURVE_fit_bad_tolerance_c	21106
PK_BCURVE_fit_bad_err_method_c	21107
PK_BCURVE_fit_bad_eval_type_c	21108
PK_BCURVE_fit_bad_break_parm_c	21109
PK_BCURVE_fit_bad_chain_c	21110
PK_BCURVE_fit_bad_fit_parm_c	21111

PK_BCURVE_fit_err_method_t

PK_BCURVE_fit_err_none_c	21090
PK_BCURVE_fit_err_parm_c	21091
PK_BCURVE_fit_err_closest_c	21092

PK_BCURVE_fit_eval_type_t

PK_BCURVE_fit_eval_user_c	21080
PK_BCURVE_fit_eval_chain_c	21081

PK_BCURVE_form_t (8650-8699)

PK_BCURVE_form_unset_c	8650
PK_BCURVE_form_arbitrary_c	8651
PK_BCURVE_form_polyline_c	8652
PK_BCURVE_form_circular_c	8653
PK_BCURVE_form_elliptic_c	8654
PK_BCURVE_form_parabolic_c	8655
PK_BCURVE_form_hyperbolic_c	8656

PK_blend_check_t

PK_blend_check_vertex_c	18380
PK_blend_check_full_c	18381

PK_blend_check_fa_fa_t

PK_blend_check_fa_fa_no_c	21060
PK_blend_check_fa_fa_yes_c	21061

PK_blend_check_fa_t

PK_blend_check_fa_no_c	21050
PK_blend_check_fa_yes_c	21051

PK_blend_check_su_X_t

PK_blend_check_su_X_no_c	21040
PK_blend_check_su_X_not_bsurf_c	21041
PK_blend_check_su_X_yes_c	21042

PK_blend_edge_update_t

PK_blend_edge_update_0_c	22190
PK_blend_edge_update_1_c	22191

PK_blend_fault_t

PK_blend_fault_no_blend_c	18390
PK_blend_fault_no_fault_c	18391
PK_blend_fault_self_int_c	18392
PK_blend_fault_face_face_c	18393
PK_blend_fault_conic_c	18394
PK_blend_fault_tangent_c	BLCCTN
PK_blend_fault_singularity_c	BLCCSN
PK_blend_fault_obsolete_c	BLCCOT
PK_blend_fault_vertex_c	BLCCMX
PK_blend_fault_sheet_c	BLCCRS
PK_blend_fault_general_c	BLCCRE
PK_blend_fault_2_edge_c	BLCCTV
PK_blend_fault_chamfer_c	BLCCHM

PK_blend_fault_bsurf_c	BLCCXT
PK_blend_fault_range_c	BLCCIR
PK_blend_fault_type_c	BLCCIT
PK_blend_fault_edge_c	BLCCAB
PK_blend_fault_loop_c	BLCCOL
PK_blend_fault_overlap_c	BLCCOB
PK_blend_fault_overlap_edge_c	BLCCOU
PK_blend_fault_bad_end_c	BLCCUE
PK_blend_fault_face_c	BLCCRL
PK_blend_fault_overlap_end_c	BLCCOE
PK_blend_fault_end_c	BLCCIE
PK_blend_fault_chamfer_intsec_c	BLCCIX
PK_blend_fault_edge_intsec_c	BLCCEX
PK_blend_fault_other_edge_c	BLCCOI
PK_blend_fault_cliffedge_c	BLCCIP
PK_blend_fault_unknown_c	BLCCUN

PK_blend_group_rib_t

PK_blend_group_rib_no_c	21030
PK_blend_group_rib_by_face_c	21031
PK_blend_group_rib_by_parms_c	21032

PK_blend_identify_t

PK_blend_identify_within_c	22240
PK_blend_identify_exc_chain_c	22241
PK_blend_identify_inc_chain_c	22242
PK_blend_identify_max_chain_c	22243
PK_blend_identify_dependent_c	22244

PK_blend_output_rib_t

PK_blend_output_rib_no_c	21020
PK_blend_output_rib_on_fail_c	21021

PK_blend_output_rib_only_c	21022
PK_blend_output_rib_also_c	21023

PK_blend_ov_cliff_t

PK_blend_ov_cliff_no_c	18450
PK_blend_ov_cliff_diff_c	18451
PK_blend_ov_cliff_any_c	18452

PK_blend_ov_cliff_end_t

PK_blend_ov_cliff_end_no_c	18460
PK_blend_ov_cliff_end_yes_c	18461

PK_blend_ov_notch_t

PK_blend_ov_notch_no_c	18470
PK_blend_ov_notch_yes_c	18471

PK_blend_ov_smooth_t

PK_blend_ov_smooth_no_c	18440
PK_blend_ov_smooth_diff_c	18441
PK_blend_ov_smooth_any_c	18442

PK_blend_propagate_t

PK_blend_propagate_no_c	18410
PK_blend_propagate_yes_c	18411

PK_blend_render_rib_t

PK_blend_render_rib_no_c	18430
PK_blend_render_rib_yes_c	18431

PK_blend_transfer_topol_t

PK_blend_transfer_topol_no_c	21770
PK_blend_transfer_topol_yes_c	21771

PK_blend_trim_t

PK_blend_trim_no_c	FXFTNT
PK_blend_trim_short_c	FXFTST
PK_blend_trim_long_c	FXFTLT
PK_blend_trim_to_walls_c	FXFTTB

PK_blend_type_t

PK_blend_type_no_c	18400
PK_blend_type_ball_constant_c	18401
PK_blend_type_ball_variable_c	18402
PK_blend_type_chamfer_c	18403

PK_blend_vary_t

PK_blend_vary_smooth_c	18420
PK_blend_vary_linear_c	18421

PK_blend_update_t

PK_blend_update_0_c	21070
PK_blend_update_1_c	21071

PK_blend_walls_t

PK_blend_walls_trim_no_c	18480
PK_blend_walls_trim_both_c	FXFTTW
PK_blend_walls_attach_c	FXFTAT
PK_blend_walls_solid_c	FXFTSO

PK_blend_xs_plane_t

PK_blend_xs_rolling_ball_c	18490
PK_blend_xs_disc_c	18491
PK_blend_xs_isoparameter_c	18492

PK_blend_xs_shape_t

PK_blend_xs_shape_unset_c	22200
PK_blend_xs_shape_conic_c	22201
PK_blend_xs_shape_g2_c	22202
PK_blend_xs_shape_chamfer_c	22203

PK_BODY_curve_nmnl_t

PK_BODY_curve_nmnl_no_c	21200
PK_BODY_curve_nmnl_yes_c	21201

PK_BODY_knit_pattern_t

PK_BODY_knit_completed_c	RTKNOK
PK_BODY_knit_not_completed_c	RTKNIN

PK_BODY_loft_clamp_type_t

PK_BODY_loft_clamp_no_c	21540
PK_BODY_loft_clamp_vector_c	21541
PK_BODY_loft_clamp_face_c	21542
PK_BODY_loft_clamp_planar_c	21543

PK_BODY_loft_curvature_t

PK_BODY_curv_natural_c	21530
PK_BODY_curv_unconstrained_no_c	21531

PK_BODY_loft_fault_t

PK_BODY_loft_ok_c	21550
PK_BODY_loft_failure_c	21551
PK_BODY_loft_bad_profile_c	21552
PK_BODY_loft_surf_failure_c	21553
PK_BODY_loft_check_failure_c	21554
PK_BODY_loft_geom_self_int_c	21555
PK_BODY_loft_topol_self_int_c	21556
PK_BODY_loft_geom_degen_c	21557
PK_BODY_loft_small_profile_c	21558

PK_BODY_owb_gap_fill_t

PK_BODY_owb_gap_fill_round_c	21220
PK_BODY_owb_gap_fill_linear_c	21221
PK_BODY_owb_gap_fill_natural_c	21222

PK_BODY_pick_method_t

PK_BODY_pick_radial_c	20900
PK_BODY_pick_axial_c	20901
PK_BODY_pick_ratio_c	20902

PK_BODY_simplify_t

PK_BODY_simplify_yes_c	21520
PK_BODY_simplify_no_c	21521

PK_BODY_sweep_alignment_t

PK_BODY_sweep_align_normal_c	21560
PK_BODY_sweep_align_parallel_c	21561

PK_BODY_sweep_fault_t

PK_BODY_sweep_ok_c	21580
PK_BODY_sweep_failure_c	21581
PK_BODY_sweep_bad_path_c	21582
PK_BODY_sweep_bad_profile_c	21583
PK_BODY_sweep_surf_failure_c	21584
PK_BODY_sweep_curve_failure_c	21585
PK_BODY_sweep_check_failure_c	21586
PK_BODY_sweep_geom_self_int_c	21587
PK_BODY_sweep_topol_self_int_c	21588
PK_BODY_sweep_geom_degen_c	21589
PK_BODY_sweep_small_profile_c	21590

PK_BODY_sweep_law_type_t

PK_BODY_sweep_law_none_c	21570
PK_BODY_sweep_law_discrete_c	21571
PK_BODY_sweep_law_curve_c	21572

PK_BODY_type_t (ENBY00)

PK_BODY_type_minimum_c	ENBYMN
PK_BODY_type_wire_c	ENBYWR
PK_BODY_type_sheet_c	ENBYSH
PK_BODY_type_solid_c	ENBYSO
PK_BODY_type_general_c	ENBYGN
PK_BODY_type_acorn_c	5606

PK_BODY_fault_t

PK_BODY_fault_no_fault_c	RTTOOK
PK_BODY_fault_bad_body_c	RTTOBB
PK_BODY_fault_unknown_c	RTTODE
PK_BODY_fault_unknown_child_c	RTTOUC
PK_BODY_fault_duplicate_child_c	RTTODC

PK_BODY_fault_wrong_child_c	RTTOWC
PK_BODY_fault_missing_child_c	RTTOFC
PK_BODY_fault_extra_child_c	RTTOMC
PK_BODY_fault_wrong_parent_c	RTTOWP
PK_BODY_fault_missing_parent_c	RTTOFP
PK_BODY_fault_extra_parent_c	RTTOPM
PK_BODY_fault_bad_wire_c	RTTODW
PK_BODY_fault_bad_loop_c	RTTOIL
PK_BODY_fault_bad_shells_c	RTTOCS
PK_BODY_fault_disjoint_shell_c	RTTODS
PK_BODY_fault_non_manifold_c	RTTONM

PK_BODY_sewing_problem_t

PK_BODY_sewing_overlapping_c	18060
PK_BODY_sewing_non_manifold_c	18061
PK_BODY_sewing_non_oriented_c	18062
PK_BODY_sewing_unspecified_c	18063

PK_BODY_sewing_removal_t

PK_BODY_sewing_remove_none_c	18076
PK_BODY_sewing_remove_poss_c	18077
PK_BODY_sewing_remove_cert_c	18078

PK_BODY_sewing_type_t

PK_BODY_sewing_solid_c	18070
PK_BODY_sewing_sheet_c	18071
PK_BODY_sewing_general_c	18072
PK_BODY_sewing_any_c	18073

PK_BODY_sweep_fair_t

PK_BODY_sweep_fair_no_c	22210
PK_BODY_sweep_fair_twist_c	22211

PK_BODY_topology_t

PK_BODY_topology_minimal_c	21830
PK_BODY_topology_columns_c	21831
PK_BODY_topology_grid_c	21832

PK_boolean_check_fa_t

PK_boolean_check_fa_no_c	21800
PK_boolean_check_fa_yes_c	21801

PK_boolean_fence_t

PK_boolean_fence_front_c	18210
PK_boolean_fence_back_c	18211
PK_boolean_fence_none_c	18212

PK_boolean_function_t (BOOP00)

PK_boolean_intersect_c	BOOPIN
PK_boolean_subtract_c	BOOPSU
PK_boolean_unite_c	BOOPUN

PK_boolean_match_t

PK_boolean_match_exact_c	18250
PK_boolean_match_contains_c	18251
PK_boolean_match_overlap_c	18252
PK_boolean_match_imprinted_c	18253

PK_boolean_match_style_t

PK_boolean_match_style_basic_c	21990
PK_boolean_match_style_relax_c	21991

PK_boolean_region_t (BOOP00)

PK_boolean_include_c	BOOPIC
PK_boolean_exclude_c	BOOPEC
PK_boolean_off_c	15949

PK_boolean_report_t

PK_boolean_report_failure_c	21600
PK_boolean_report_opposed_sheets_c	21601
PK_boolean_report_solid_has_void_c	21602
PK_boolean_report_partial_coi_c	21603
PK_boolean_report_t_sheet_c	21604
PK_boolean_report_non_manifold_c	21605
PK_boolean_report_invalid_match_c	21606
PK_boolean_report_imprint_fail_c	21607
PK_boolean_report_intersect_fail_c	21608
PK_boolean_report_Invalid_face_c	21609

PK_boolean_result_t

PK_boolean_result_success_c	21650
PK_boolean_result_no_clash_c	21651
PK_boolean_result_no_effect_c	21652
PK_boolean_result_failed_c	21653

PK_boolean_select_t

PK_boolean_select_specific_c	21000
PK_boolean_select_adjacent_c	21001
PK_boolean_select_propagate_c	21002

PK_bound_t

PK_bound_distance_c	21850
PK_bound_sheet_c	21851

PK_bound_surf_c	21852
PK_bound_face_c	21853
PK_bound_body_c	21854

PK_BSURF_form_t

PK_BSURF_form_unset_c	8700
PK_BSURF_form_arbitrary_c	8701
PK_BSURF_form_planar_c	8702
PK_BSURF_form_cylindrical_c	8703
PK_BSURF_form_conical_c	8704
PK_BSURF_form_spherical_c	8705
PK_BSURF_form_toroidal_c	8706
PK_BSURF_form_revolved_c	8707
PK_BSURF_form_ruled_c	8708
PK_BSURF_form_gen_cone_c	8709
PK_BSURF_form_quadric_c	8710
PK_BSURF_form_swept_c	8711

PK_check_geom_t

PK_check_geom_no_c	18280
PK_check_geom_basic_c	18281
PK_check_geom_lazy_c	18282
PK_check_geom_full_c	18283
PK_check_geom_yes_c	18284

PK_check_bgeom_t

PK_check_bgeom_no_c	18290
PK_check_bgeom_yes_c	18291

PK_check_corrupt_t

PK_check_corrupt_no_c	18360
PK_check_corrupt_yes_c	18361

PK_check_top_geo_t

PK_check_top_geo_no_c	18300
PK_check_top_geo_edge_c	18301
PK_check_top_geo_face_c	18302
PK_check_top_geo_yes_c	18303

PK_check_size_box_t

PK_check_size_box_no_c	18310
PK_check_size_box_yes_c	18311

PK_check_fa_X_t

PK_check_fa_X_no_c	18320
PK_check_fa_X_yes_c	18321

PK_check_loops_t

PK_check_loops_no_c	18330
PK_check_loops_yes_c	18331

PK_check_fa_fa_t

PK_check_fa_fa_no_c	18340
PK_check_fa_fa_yes_c	18341

PK_check_sh_t

PK_check_sh_no_c	18350
PK_check_sh_yes_c	18351

PK_check_state_t

PK_ATTDEF_state_bad_name_c	RTSTAN
PK_ATTRIB_state_bad_string_c	RTSTAS
PK_BODY_state_corrupt_c	RTSTCR
PK_BODY_state_invalid_ident_c	RTSTID

PK_BODY_state_inside_out_c	RTSTNG
PK_BODY_state_bad_regions_c	RTSTRS
PK_BODY_state_ok_c	RTSTOK
PK_CURVE_state_inconsistent_c	RTTSCI
PK_EDGE_state_open_c	RTSTOC
PK_EDGE_state_open_nmnl_c	RTSTON
PK_EDGE_state_bad_vertex_c	RTSTVC
PK_EDGE_state_bad_vertex_nmnl_c	RTSTVN
PK_EDGE_state_reversed_c	RTSTER
PK_EDGE_state_reversed_nmnl_c	RTSTRN
PK_EDGE_state_bad_spcurve_c	RTSTSP
PK_EDGE_state_bad_sp_nmnl_c	RTSTSN
PK_EDGE_state_vertices_touch_c	RTSTVT
PK_EDGE_state_bad_face_order_c	RTSTFO
PK_EDGE_state_bad_order_c	RTTSEO
PK_EDGE_state_bad_wire_ed_ed_c	RTSTEE
PK_ENTITY_state_invalid_c	RTSTIG
PK_FACE_state_bad_vertex_c	RTSTVS
PK_FACE_state_bad_edge_c	RTSTES
PK_FACE_state_bad_edge_order_c	RTSTEO
PK_FACE_state_no_vtx_at_sing_c	RTSTMV
PK_FACE_state_bad_loops_c	RTSTLC
PK_FACE_state_self_int_c	RTSTSX
PK_FACE_state_bad_wire_fa_ed_c	RTSTEF
PK_FACE_state_check_fail_c	RTSTFC
PK_FACE_state_bad_face_face_c	RTSTFF
PK_FACE_state_redundant_c	RTTSFR
PK_GEOM_state_self_int_c	RTSTGX
PK_GEOM_state_degenerate_c	RTSTDG
PK_LOOP_state_invalid_c	RTTSLI
PK_REGION_state_bad_shells_c	RTSTSH
PK_SHELL_state_bad_topol_geom_c	RTSTSG
PK_SHELL_state_bad_sh_sh_c	RTSTAC

PK_TOPOL_state_not_G1_c	RTSTGC
PK_TOPOL_state_size_box_c	RTSTBX
PK_TOPOL_state_check_fail_c	RTSTCF
PK_TOPOL_state_no_geom_c	RTSTMG

PK_CLASS_t

PK_CLASS_null	500
PK_CLASS_class	501

special classes

PK_CLASS_entity	TYEN00
PK_CLASS_primitive	502
PK_CLASS_error	503
PK_CLASS_session	504
PK_CLASS_memory	505
PK_CLASS_mark	506
PK_CLASS_pmark	507
PK_CLASS_partition	508
PK_CLASS_bb	509

primitive classes

PK_CLASS_int	510
PK_CLASS_double	511
PK_CLASS_char	512
PK_CLASS_string	513
PK_CLASS_logical	514
PK_CLASS_vector	515
PK_CLASS_interval	516
PK_CLASS_box	517
PK_CLASS_uvbox	518
PK_CLASS_uv	519
PK_CLASS_pointer	520

PK_CLASS_vector1	521
PK_CLASS_size	522

entity classes

PK_CLASS_attrib	TYENAD
PK_CLASS_attdef	TYADAD
PK_CLASS_group	TYADFE
PK_CLASS_transf	TYGETF
PK_CLASS_KI_list	TYADLI

topol

PK_CLASS_topol	TYENTO
PK_CLASS_part	5012
PK_CLASS_assembly	TYTOAS
PK_CLASS_body	TYTOBY
PK_CLASS_instance	TYTOIN
PK_CLASS_region	TYTORG
PK_CLASS_shell	TYTOSH
PK_CLASS_face	TYTOFA
PK_CLASS_loop	TYTOLO
PK_CLASS_edge	TYTOED
PK_CLASS_fin	TYTOFN
PK_CLASS_vertex	TYTOVX

geom

PK_CLASS_geom	TYENGE

surf

PK_CLASS_surf	TYGESU
PK_CLASS_plane	TYSUPL
PK_CLASS_cyl	TYSUCY
PK_CLASS_cone	TYSUCO

PK_CLASS_sphere	TYSUSP
PK_CLASS_torus	TYSUTO
PK_CLASS_bsurf	TYSUPA
PK_CLASS_offset	TYSUOF
PK_CLASS_fsurf	TYSUTG
PK_CLASS_swept	TYSUSE
PK_CLASS_spun	TYSUSU
PK_CLASS_blendsf	TYSUBL

curve

PK_CLASS_curve	TYGECU
PK_CLASS_line	TYCUST
PK_CLASS_circle	TYCUCI
PK_CLASS_ellipse	TYCUEL
PK_CLASS_bcurve	TYCUPA
PK_CLASS_icurve	TYCUIN
PK_CLASS_fcurve	TYCUFG
PK_CLASS_spcurve	TYCUSP
PK_CLASS_trcurve	TYCUTR
PK_CLASS_cpcurve	TYCUCP

point

PK_CLASS_point	TYPTCA

nabox

PK_CLASS_nabox	2005

PK_comparison_t

PK_comparison_always_c	22220
PK_comparison_less_c	22221
PK_comparison_equal_c	22222
PK_comparison_greater_c	22223

PK_comparison_between_c	22224
PK_comparison_outside_c	22225
PK_comparison_never_c	22226

PK_convexity_t (8600-8649)

PK_convexity_unset_c	8600
PK_convexity_arbitrary_c	8601
PK_convexity_convex_c	8602
PK_convexity_concave_c	8603

PK_CURVE_approx_t

PK_CURVE_approx_arclength_c	22120
PK_CURVE_approx_even_c	22129

PK_DEBUG_diff_t

PK_DEBUG_diff_n_shells_c	21940
PK_DEBUG_diff_n_faces_c	21941
PK_DEBUG_diff_n_loops_c	21942
PK_DEBUG_diff_n_acc_edges_c	21943
PK_DEBUG_diff_n_tol_edges_c	21944
PK_DEBUG_diff_n_fins_c	21945
PK_DEBUG_diff_n_acc_vxs_c	21946
PK_DEBUG_diff_n_tol_vxs_c	21947
PK_DEBUG_diff_n_vxs_c	21948
PK_DEBUG_diff_n_cht_pts_c	21949
PK_DEBUG_diff_surf_dev_c	21950
PK_DEBUG_diff_curve_dev_c	21951
PK_DEBUG_diff_vx_dev_c	21952
PK_DEBUG_diff_surf_class_c	21953
PK_DEBUG_diff_curve_class_c	21954
PK_DEBUG_diff_edge_tol_c	21955
PK_DEBUG_diff_vx_tol_c	21956

PK_DEBUG_diff_face_sense_c	21957
PK_DEBUG_diff_surf_sense_c	21958
PK_DEBUG_diff_curve_sense_c	21959
PK_DEBUG_diff_vx_missing_c	21960
PK_DEBUG_diff_face_match_c	21961
PK_DEBUG_diff_fin_match_c	21962
PK_DEBUG_diff_vx_match_c	21963

PK_DEBUG_diff_type_t

PK_DEBUG_diff_type_all_c	21900
PK_DEBUG_diff_type_nums_c	21901
PK_DEBUG_diff_type_devs_c	21902
PK_DEBUG_diff_type_class_c	21903
PK_DEBUG_diff_type_tols_c	21904
PK_DEBUG_diff_type_sense_c	21905
PK_DEBUG_diff_type_miss_c	21906
PK_DEBUG_diff_type_non_match_c	21907

PK_DEBUG_global_res_t

PK_DEBUG_global_res_diffs_c	21920
PK_DEBUG_global_res_no_diffs_c	21921

PK_DEBUG_local_res_t

PK_DEBUG_local_res_diffs_c	21930
PK_DEBUG_local_res_no_diffs_c	21931

PK_DEBUG_global_type_t

PK_DEBUG_global_shells_c	21940
PK_DEBUG_global_faces_c	21941
PK_DEBUG_global_acc_edges_c	21942
PK_DEBUG_global_tol_edges_c	21943

PK_DEBUG_global_acc_verts_c	21944
PK_DEBUG_global_tol_verts_c	21945

PK_DEBUG_local_type_t

PK_DEBUG_local_num_loops_c	21960
PK_DEBUG_local_num_hes_c	21961
PK_DEBUG_local_num_vxs_c	21962
PK_DEBUG_local_num_cht_pts_c	21963
PK_DEBUG_local_surf_dev_c	21964
PK_DEBUG_local_curve_dev_c	21965
PK_DEBUG_local_vx_dev_c	21966
PK_DEBUG_local_surf_class_c	21967
PK_DEBUG_local_curve_class_c	21968
PK_DEBUG_local_edge_tol_c	21969
PK_DEBUG_local_vx_tol_c	21970
PK_DEBUG_local_face_sense_c	21971
PK_DEBUG_local_surf_sense_c	21972
PK_DEBUG_local_curve_sense_c	21973
PK_DEBUG_local_vx_missing_c	21974
PK_DEBUG_local_face_match_c	21975
PK_DEBUG_local_he_match_c	21976
PK_DEBUG_local_vx_match_c	21977

PK_detail_t

PK_detail_any_c	22250
PK_detail_rubber_c	22251
PK_detail_hole_cyl_c	22252
PK_detail_hole_cyl_through_c	22253
PK_detail_hole_cyl_blind_c	22254
PK_detail_hole_cyl_closed_c	22255
PK_detail_blend_rb_const_r_c	22256

PK_EDGE_fins_type_t (3700-3749)

PK_EDGE_type_wireframe_c	3700
PK_EDGE_type_laminar_c	3701
PK_EDGE_type_normal_c	3702
PK_EDGE_type_general_c	3703

PK_EDGE_vertex_type_t (3650-3699)

PK_EDGE_type_open_c	3650
PK_EDGE_type_closed_c	3651
PK_EDGE_type_ring_c	3652

PK_enclosure_t (ENCL00)

PK_enclosure_inside_c	ENCLIN
PK_enclosure_outside_c	ENCLOU
PK_enclosure_on_c	ENCLON

PK_ENTITY_null

PK_ENTITY_null	0

PK_ERROR_severity_t

PK_ERROR_none	0
PK_ERROR_mild	1
PK_ERROR_serious	2
PK_ERROR_fatal	3

PK_FACE_change_type_t

PK_FACE_change_type_none_c	22060
PK_FACE_change_type_offset_c	22061
PK_FACE_change_type_taper_c	22062
PK_FACE_change_type_transform_c	22063

PK_FACE_change_type_replace_c	22064
PK_FACE_change_type_blend_c	22065

PK_FACE_coi_t

PK_FACE_coi_yes_c	21880
PK_FACE_coi_yes_reversed_c	21881
PK_FACE_coi_coi_no_topol_c	21882
PK_FACE_coi_coi_no_bound_1_c	21883
PK_FACE_coi_no_bound_2_c	21884
PK_FACE_coi_no_face_1_c	21885
PK_FACE_coi_no_face_2_c	21886
PK_FACE_coi_no_rubber_c	21887

PK_FACE_heal_t

PK_FACE_heal_no_c	18080
PK_FACE_heal_cap_c	18081
PK_FACE_heal_grow_from_child_c	18082
PK_FACE_heal_grow_from_parent_c	18083
PK_FACE_heal_shrink_c	18084
PK_FACE_heal_yes_c	18085

PK_FACE_trim_surf_t

PK_FACE_trim_surf_own_c	18370
PK_FACE_trim_surf_bsurf_c	SROPBS

PK_FACE_trim_confine_t

PK_FACE_trim_confine_no_c	SROPCN
PK_FACE_trim_confine_yes_c	SROPCY
PK_FACE_trim_confine_closed_c	SROPCC
PK_FACE_trim_confine_periph_c	SROPCP

PK_FACE_trim_degen_t

PK_FACE_trim_degen_no_c	SROPED
PK_FACE_trim_degen_yes_c	SROPID

PK_facet_cull_t

PK_facet_cull_no_c	20560
PK_facet_cull_back_c	20561

PK_facet_degen_t

PK_facet_degen_multiple_vxs_c	20580
PK_facet_degen_single_vx_c	20581
PK_facet_degen_average_parms_c	20582

PK_facet_density_t

PK_facet_density_no_view_c	20540
PK_facet_density_use_view_c	20541

PK_facet_fault_t

PK_facet_fault_face_c	20700
PK_facet_fault_facet_c	20701
PK_facet_fault_vertex_point_c	20702
PK_facet_fault_vertex_normal_c	20703
PK_facet_fault_vertex_param_c	20704
PK_facet_fault_thin_face_c	20705
PK_facet_fault_body_c	20706

PK_facet_go_edges_t

PK_facet_go_edges_no_c	20640
PK_facet_go_edges_yes_c	20641

PK_facet_go_interleaved_t

PK_facet_go_interleaved_no_c	20680
PK_facet_go_interleaved_yes_c	20681

PK_facet_go_normals_t

PK_facet_go_normals_no_c	20600
PK_facet_go_normals_yes_c	20601

PK_facet_go_parameters_t

PK_facet_go_parameters_no_c	20620
PK_facet_go_parameters_d0_c	20621
PK_facet_go_parameters_d1_c	20622
PK_facet_go_parameters_d2_c	20623

PK_facet_go_strips_t

PK_facet_go_strips_no_c	20660
PK_facet_go_strips_yes_c	20661

PK_facet_ignore_t

PK_facet_ignore_no_c	22111
PK_facet_ignore_absolute_c	22112
PK_facet_ignore_ratio_c	22113
PK_facet_ignore_body_ratio_c	22114

PK_facet_ignore_scope_t

PK_facet_ignore_scope_local_c	22130
PK_facet_ignore_scope_global_c	22131

PK_facet_match_t

PK_facet_match_no_c	20520
PK_facet_match_geom_c	20521

PK_facet_match_topol_c	20522
PK_facet_match_trimmed_c	20523

PK_facet_shape_t

PK_facet_shape_any_c	20500
PK_facet_shape_cut_c	20501
PK_facet_shape_convex_c	20502

PK_facet_wire_edges_t

PK_facet_wire_edges_no_c	22140
PK_facet_wire_edges_yes_c	22141

PK_fill_hole_fault_t

PK_fill_hole_ok_c	22080
PK_fill_hole_bad_edge_c	22081
PK_fill_hole_duplicate_c	22082
PK_fill_hole_vertex_c	22083
PK_fill_hole_too_many_loops_c	22084
PK_fill_hole_gap_c	22085
PK_fill_hole_no_edge_on_target_c	22086
PK_fill_hole_cant_match_body_c	22087
PK_fill_hole_cant_match_edge_c	22088
PK_fill_hole_too_small_c	22089
PK_fill_hole_unknown_c	22090
PK_fill_hole_face_c	22091
PK_fill_hole_face_face_c	22092
PK_fill_hole_not_smooth_c	22093
PK_fill_hole_too_complex_c	22094

PK_fill_hole_method_t

PK_fill_hole_trim_to_hole_c	22100

PK_fill_hole_trim_to_sheet_c	22101
PK_fill_hole_create_patch_c	22102

PK_FIN_type_t (3600-3649)

PK_FIN_type_wire_c	3600
PK_FIN_type_biwire_c	3601
PK_FIN_type_normal_c	3602

PK_fxf_fault_t

PK_fxf_fault_no_fault_c	FXFEOK
PK_fxf_fault_sheet_c	FXFEST
PK_fxf_fault_unknown_c	FXFEER
PK_fxf_fault_insufficient_c	FXFEID
PK_fxf_fault_inconsistent_c	FXFEXD
PK_fxf_fault_wall_c	FXFEIF
PK_fxf_fault_range_c	FXFEIR
PK_fxf_fault_cliff_c	FXFEIC
PK_fxf_fault_thl_c	FXFEIH
PK_fxf_fault_chl_c	FXFECL
PK_fxf_fault_curved_c	FXFEFC
PK_fxf_fault_small_c	FXFERS
PK_fxf_fault_large_c	FXFERL
PK_fxf_fault_left_c	FXFELN
PK_fxf_fault_right_c	FXFERN
PK_fxf_fault_both_c	FXFEBN
PK_fxf_fault_sheet_clash_c	FXFESC
PK_fxf_fault_wall_clash_c	FXFEWC
PK_fxf_fault_face_face_c	FXFEFF
PK_fxf_fault_self_int_c	FXFEGX
PK_fxf_fault_rho_value_c	FXFERV
PK_fxf_fault_asymmetric_c	FXFEAR
PK_fxf_fault_bad_spine_c	FXFEIS
PK_fxf_fault_bad_ribs_c	17474

PK_HAND_t (32760-32769)

PK_HAND_left_c	32760
PK_HAND_right_c	32761

PK_intersect_curve_t

PK_intersect_curve_simple_c	SICLSI
PK_intersect_curve_tangent_c	SICLTG

PK_intersect_fc_t

PK_intersect_fc_simple_c	CFCLSI
PK_intersect_fc_tangent_c	CFCLTG
PK_intersect_fc_out_in_c	CFCLEF
PK_intersect_fc_in_out_c	CFCLLF
PK_intersect_fc_out_coi_c	CFCLEB
PK_intersect_fc_coi_out_c	CFCLLB
PK_intersect_fc_coi_in_c	CFCLEI
PK_intersect_fc_in_coi_c	CFCLLI
PK_intersect_fc_in_tangent_c	CFCLTI
PK_intersect_fc_out_tangent_c	CFCLTO
PK_intersect_fc_in_c	CFCLUC
PK_intersect_fc_start_c	CFCLSC
PK_intersect_fc_end_c	CFCLEC

PK_intersect_vector_t

PK_intersect_vector_simple_c	CICLSI
PK_intersect_vector_start_c	CICLSC
PK_intersect_vector_end_c	CICLEC
PK_intersect_vector_tangent_c	CICLTG

PK_knot_type_t (8500-8549)

PK_knot_unset_c	8500
PK_knot_non_uniform_c	8501
PK_knot_uniform_c	8502
PK_knot_quasi_uniform_c	8503
PK_knot_piecewise_bezier_c	8504
PK_knot_bezier_ends_c	8505
PK_knot_smooth_seam_c	8506

PK_local_change_topol_t

PK_local_change_topol_no_c	21420
PK_local_change_topol_yes_c	21421

PK_local_check_t

PK_local_check_no_c	18100
PK_local_check_ok_c	18101
PK_local_check_negated_c	18102
PK_local_check_failed_c	18103

PK_local_check_fa_fa_t

PK_local_check_fa_fa_no_c	21780
PK_local_check_fa_fa_yes_c	21781

PK_local_check_su_X_t

PK_local_check_su_X_no_c	21400
PK_local_check_su_X_yes_c	21401

PK_local_check_topol_t

PK_local_check_topol_no_c	21410
PK_local_check_topol_yes_c	21411

PK_local_status_t

PK_local_status_ok_c	21450
PK_local_status_nocheck_c	21451
PK_local_status_fail_c	21452
PK_local_status_cant_get_pt_c	21453
PK_local_status_cant_get_cu_c	21454
PK_local_status_cant_get_su_c	21455
PK_local_status_cant_offset_c	21456
PK_local_status_side_cu_fail_c	21457
PK_local_status_side_su_fail_c	21458
PK_local_status_not_same_by_c	21459
PK_local_status_fa_fail_c	21460
PK_local_status_fa_fa_fail_c	21461
PK_local_status_ed_remains_c	21462
PK_local_status_point_contact_c	21463
PK_local_status_bad_reference_c	21464
PK_local_status_not_supported_c	21465

PK_LOGICAL_t

PK_LOGICAL_true	01
PK_LOGICAL_false	00

PK_LOOP_type_t

PK_LOOP_type_hole_c	ENLOHO
PK_LOOP_type_peripheral_c	ENLOPE
PK_LOOP_type_other_c	ENLONA
PK_LOOP_type_vertex_c	5410
PK_LOOP_type_wire_c	5411
PK_LOOP_type_outer_c	5412
PK_LOOP_type_inner_c	5413
PK_LOOP_type_winding_c	5414
PK_LOOP_type_inner_sing_c	5415

PK_LOOP_type_likely_outer_c	5416
PK_LOOP_type_likely_inner_c	5417
PK_LOOP_type_unclear_c	5418
PK_LOOP_type_error_c	5419

PK_MARK_null

PK_MARK_null	0

PK_mass_t

PK_mass_no_c	MAOPNA
PK_mass_mass_c	MAOPAM
PK_mass_c_of_g_c	MAOPCG
PK_mass_m_of_i_c	MAOPIN

PK_mass_periphery_t

PK_mass_periphery_no_c	MAOPNP
PK_mass_periphery_yes_c	MAOPPE

PK_mass_bound_t

PK_mass_bound_no_c	MAOPNE
PK_mass_bound_modulus_c	MAOPEM
PK_mass_bound_interval_c	MAOPEI

PK_neutral_error_t (3800-3849)

PK_neutral_success_c	3800
PK_neutral_illegal_input_c	3801
PK_neutral_numerical_failure_c	3802
PK_neutral_unknown_config_c	3803
PK_neutral_imprint_failure_c	3804

PK_offset_method_t

PK_offset_method_accurate_c	22150
PK_offset_method_approximate_c	22151

PK_offset_step_t

PK_offset_step_no_c	22310
PK_offset_step_yes_c	22311

PK_outline_project_t

PK_outline_project_no_c	22170
PK_outline_project_plane_c	22171

PK_PARAM_bound_t

PK_PARAM_bound_infinite_c	18000
PK_PARAM_bound_extendable_c	18001
PK_PARAM_bound_bound_c	18002
PK_PARAM_bound_closed_c	18003
PK_PARAM_bound_degenerate_c	18004

PK_PARAM_degen_t

PK_PARAM_degen_no_c	18550
PK_PARAM_degen_vector_c	18551
PK_PARAM_degen_bcurve_c	18552

PK_PARAM_direction_t

PK_PARAM_direction_u_c	22070
PK_PARAM_direction_v_c	22071

PK_PARAM_end_t

PK_PARAM_end_periodic_c	18510
PK_PARAM_end_natural_c	18511

PK_PARAM_end_clamped_vector_c	18512
PK_PARAM_end_bcurve_c	18513

PK_PARAM_form_t

PK_PARAM_form_linear_c	18040
PK_PARAM_form_circular_c	18041
PK_PARAM_form_any_c	18042

PK_PARAM_knot_t

PK_PARAM_knot_auto_c	18520
PK_PARAM_knot_defined_c	18521

PK_PARAM_periodic_t

PK_PARAM_periodic_no_c	18020
PK_PARAM_periodic_yes_c	18021
PK_PARAM_periodic_seamed_c	18022

PK_PARAM_twist_t

PK_PARAM_twist_no_c	18530
PK_PARAM_twist_yes_c	18531

PK_PARTITION_null

PK_PARTITION_null	0

PK_PARTITION_xmt_deltas_t

PK_PARTITION_xmt_deltas_none_c	18230
PK_PARTITION_xmt_deltas_all_c	18231
PK_PARTITION_xmt_deltas_main_c	18232

PK_PARTITION_rcv_deltas_t

PK_PARTITION_rcv_deltas_no_c	18240
PK_PARTITION_rcv_deltas_later_c	18241

PK_pattern_check_fa_fa_t

PK_pattern_check_fa_fa_no_c	21300
PK_pattern_check_fa_fa_yes_c	21301

PK_pattern_check_loops_t

PK_pattern_check_loops_no_c	21310
PK_pattern_check_loops_yes_c	21311

PK_pattern_result_t

PK_pattern_result_ok_c	21360
PK_pattern_result_partial_c	21361
PK_pattern_result_fail_c	21362
PK_pattern_result_mixed_faces_c	21363
PK_pattern_result_instanced_c	21364
PK_pattern_result_invalid_set_c	21365
PK_pattern_result_miss_geom_c	21366

PK_pattern_status_t

PK_pattern_status_ok_c	21320
PK_pattern_status_fail_c	21321
PK_pattern_status_outside_fa_c	21322
PK_pattern_status_outside_reg_c	21323
PK_pattern_status_tf_failed_c	21324
PK_pattern_status_face_clash_c	21325

PK_PMARK_null

PK_PMARK_null	0

PK_pick_approximate_t

PK_pick_approximate_no_c	22180
PK_pick_approximate_yes_c	22181

PK_piecewise_rep_t

PK_piecewise_rep_bezier_c	18500
PK_piecewise_rep_polynomial_c	18501
PK_piecewise_rep_hermite_c	18502
PK_piecewise_rep_taylor_c	18503

PK_range_guess_type_t

PK_range_guess_no_c	18260
PK_range_guess_param_c	18261
PK_range_guess_vector_c	18262

PK_range_result_t

PK_range_result_found_c	18270
PK_range_result_lower_c	18271
PK_range_result_upper_c	18272

PK_render_bcurve_t

PK_render_bcurve_polyline_c	20400
PK_render_bcurve_bezier_c	20401
PK_render_bcurve_nurbs_c	20402

PK_render_boundary_t

PK_render_boundary_no_c	20220
PK_render_boundary_yes_c	20221

PK_render_drafting_t

PK_render_drafting_no_c	20450
PK_render_drafting_yes_c	20451

PK_render_edge_t

PK_render_edge_no_c	20100
PK_render_edge_yes_c	20101

PK_render_hierarch_t

PK_render_hierarch_no_c	20380
PK_render_hierarch_no_geom_c	20381
PK_render_hierarch_yes_c	20382
PK_render_hierarch_param_c	20383

PK_render_ignore_t

PK_render_ignore_no_c	21820
PK_render_ignore_absolute_c	21821
PK_render_ignore_ratio_c	21822
PK_render_ignore_body_ratio_c	21823

PK_render_internal_t

PK_render_internal_no_c	20340
PK_render_internal_yes_c	20341

PK_render_memory_target_t

PK_render_memory_target_no_c	22160
PK_render_memory_target_yes_c	22161

PK_render_overlap_t

PK_render_overlap_no_c	21871
PK_render_overlap_yes_c	21872

PK_render_invisible_t

PK_render_invisible_no_c	20440
PK_render_invisible_yes_c	20441

PK_render_param_t

PK_render_param_no_c	20180
PK_render_param_attrib_c	20181
PK_render_param_yes_c	20182
PK_render_param_free_c	20183

PK_render_planar_t

PK_render_planar_no_c	20140
PK_render_planar_attrib_c	20141
PK_render_planar_free_c	20142
PK_render_planar_yes_c	20143

PK_render_radial_t

PK_render_radial_no_c	20160
PK_render_radial_attrib_c	20161
PK_render_radial_yes_c	20162
PK_render_radial_free_c	20163

PK_render_region_t

PK_render_region_no_c	20360
PK_render_region_attrib_c	20361
PK_render_region_yes_c	20362

PK_render_self_hidden_t

PK_render_self_hidden_no_c	20460
PK_render_self_hidden_yes_c	20461

PK_render_silhouette_t

PK_render_silhouette_no_c	20120
PK_render_silhouette_arcs_c	20121
PK_render_silhouette_yes_c	20122

PK_render_smooth_t

PK_render_smooth_no_c	20320
PK_render_smooth_yes_c	20321
PK_render_smooth_draft_c	20322

PK_render_unfix_t

PK_render_unfix_no_c	20200
PK_render_unfix_attrib_c	20201
PK_render_unfix_yes_c	20202

PK_render_viewport_t

PK_render_viewport_no_c	20420
PK_render_viewport_yes_c	20421
PK_render_viewport_array_c	20422

PK_render_vis_t

PK_render_vis_no_c	20300
PK_render_vis_hid_c	20301
PK_render_vis_inv_c	20302
PK_render_vis_hid_draft_c	20303
PK_render_vis_inv_draft_c	20304
PK_render_vis_extended_c	20305

PK_replace_merge_t

PK_replace_merge_no_c	22300

PK_replace_merge_in_c	22301
PK_replace_merge_out_c	22302

PK_reset_prec_t

PK_reset_prec_ok_c	RTTLOK
PK_reset_prec_tangent_c	RTTLNT
PK_reset_prec_missing_geom_c	RTTLMG
PK_reset_prec_failure_c	RTTLRF

PK_section_check_fa_t

PK_section_check_fa_no_c	21810
PK_section_check_fa_yes_c	21811

PK_section_fence_t

PK_section_fence_front_c	18200
PK_section_fence_back_c	18201
PK_section_fence_both_c	18202

PK_section_report_t

PK_section_report_failure_c	21700
PK_section_report_solid_has_void_c	21701
PK_section_report_partial_coi_c	21702
PK_section_report_t_sheet_c	21703
PK_section_report_non_manifold_c	21704
PK_section_report_invalid_match_c	21705
PK_section_report_imprint_fail_c	21706
PK_section_report_intersect_fail_c	21707
PK_section_report_invalid_face_c	21708

PK_section_result_t

PK_section_result_success_c	21750
PK_section_result_no_clash_c	21751
PK_section_result_no_effect_c	21752
PK_section_result_failed_c	21753

PK_self_intersect_t (8550-8599)

PK_self_intersect_unset_c	8550
PK_self_intersect_false_c	8551
PK_self_intersect_true_c	8552

PK_SHELL_type_t (3500-3549)

PK_SHELL_type_acorn_c	3500
PK_SHELL_type_wireframe_c	3501
PK_SHELL_type_wireframe_c	3502
PK_SHELL_type_mixed_c	3503

PK_SHELL_sign_t (3550-3599)

PK_SHELL_sign_positive_c	3550
PK_SHELL_sign_negative_c	3551
PK_SHELL_sign_open_c	3552

PK_step_surf_t

PK_step_tapered_c	21840
PK_step_normal_c	21841

PK_SURF_extension_t

PK_SURF_extension_none_c	22030
PK_SURF_extension_point_c	22031
PK_SURF_extension_box_c	22032

PK_SURF_extension_uvbox_c	22033
PK_SURF_extension_ratio_c	22034

PK_taper_method_t

PK_taper_method_curve_c	21845
PK_taper_method_surface_c	21846
PK_taper_method_isocline_c	21847
PK_taper_method_offset_c	21848

PK_thicken_method_t

PK_thicken_method_offset_c	22040
PK_thicken_method_punch_c	22041

PK_thread_t

PK_thread_disabled_c	21010
PK_thread_per_processor_c	21011
PK_thread_absolute_c	21012

PK_TOPOL_clash_type_t

PK_TOPOL_clash_none_c	0
PK_TOPOL_clash_a_in_b_c	1
PK_TOPOL_clash_b_in_a_c	2
PK_TOPOL_clash_exists_c	3
PK_TOPOL_clash_abut_no_class_c	4
PK_TOPOL_clash_abut_b_in_a_c	5
PK_TOPOL_clash_abut_b_out_a_c	6
PK_TOPOL_clash_interfere_c	7

PK_TOPOL_sense_t

PK_TOPOL_sense_none_c	18540

PK_TOPOL_sense_negative_c	18541
PK_TOPOL_sense_positive_c	18542

PK_TOPOL_track_t

PK_TOPOL_track_delete_c	21500
PK_TOPOL_track_merge_c	21501
PK_TOPOL_track_create_c	21502
PK_TOPOL_track_split_c	21503
PK_TOPOL_track_derive_c	21504

PK_transmit_format_t

PK_transmit_format_text_c	18220
PK_transmit_format_binary_c	18221
PK_transmit_format_neutral_c	18222
PK_transmit_format_applio_c	18223
PK_transmit_format_xml_c	18224

PK_VERTEX_type_t

PK_VERTEX_type_isolated_c	ENVEIS
PK_VERTEX_type_spur_c	ENVESP
PK_VERTEX_type_wire_c	ENVEWR
PK_VERTEX_type_normal_c	ENVENO

PK Error Codes (Numeric)

C

C.1 Numerical List of PK Error Codes

This appendix lists all the error codes used by the PK Interface in numerical order.

C.1.1 Error Codes 0-99

PK_ERROR_no_errors	0
PK_ERROR_bad_angle	1
PK_ERROR_buffer_overflow	2
PK_ERROR_radii_both_0	3
PK_ERROR_cone_too_sharp	4
PK_ERROR_has_no_name	7
PK_ERROR_has_no_owner	8
PK_ERROR_wrong_entity	10
PK_ERROR_bad_name	11
PK_ERROR_bad_type_combn	12
PK_ERROR_not_unique	13
PK_ERROR_distance_lt_0	14
PK_ERROR_distance_le_0	15
PK_ERROR_radius_le_0	16
PK_ERROR_radius_lt_0	18
PK_ERROR_not_found	19
PK_ERROR_not_connected	20
PK_ERROR_not_an_entity	22
PK_ERROR_null_axis	25
PK_ERROR_cant_open_jrnl	27
PK_ERROR_has_parent	28
PK_ERROR_bad_index	29
PK_ERROR_bad_type	30

PK_ERROR_null_direction	31
PK_ERROR_rot_angle_eq_0	32
PK_ERROR_lt_3_sides	33
PK_ERROR_is_attached	34
PK_ERROR_dont_intersect	35
PK_ERROR_majaxi_not_perpn	36
PK_ERROR_wrong_transf	37
PK_ERROR_bad_selection_code	38
PK_ERROR_bad_value	39
PK_ERROR_sc_factor_le_0	40
PK_ERROR_su_are_coincident	41
PK_ERROR_bulletinb_is_off	42
PK_ERROR_none_mergeable	48
PK_ERROR_cant_do_tweak	50
PK_ERROR_inconsistent_geom	51
PK_ERROR_not_on_face	54
PK_ERROR_impossible_swing	55
PK_ERROR_impossible_sweep	57
PK_ERROR_key_not_found	58
PK_ERROR_not_in_same_part	59
PK_ERROR_no_geometry	61
PK_ERROR_geom_topol_mismatch	62
PK_ERROR_receive_failed	63
PK_ERROR_geom_not_needed	64
PK_ERROR_not_on_curve	67
PK_ERROR_still_referenced	68
PK_ERROR_fragment	73
PK_ERROR_cant_find_su	77
PK_ERROR_empty_list	79
PK_ERROR_not_a_list	80
PK_ERROR_mass_eq_0	82
PK_ERROR_density_le_0	85
PK_ERROR_dont_make_solid	87

PK_ERROR_missing_geom	96
PK_ERROR_attr_not_found	99

C.1.2 Error Codes 100-999

PK_ERROR_not_solid	101
PK_ERROR_corrupt_body	103
PK_ERROR_bad_geom_topol	105
PK_ERROR_negative_body	106
PK_ERROR_bad_char_string	109
PK_ERROR_bad_spec_code	110
PK_ERROR_weight_le_0	111
PK_ERROR_illegal_degeneracy	116
PK_ERROR_bad_parameter	120
PK_ERROR_discontinuous_surface	129
PK_ERROR_discontinuous_curve	131
PK_ERROR_order_lt_2	132
PK_ERROR_bad_dimension	135
PK_ERROR_su_self_intersect	141
PK_ERROR_cant_do_intersect	157
PK_ERROR_cant_fix_blends	330
PK_ERROR_bad_blend_bound	334
PK_ERROR_not_blended	335
PK_ERROR_blend_didnt_check	336
PK_ERROR_bad_request_code	350
PK_ERROR_wrong_entity_in_array	357
PK_ERROR_not_same_length	359
PK_ERROR_bad_view_mx	360
PK_ERROR_bad_pixel_map	361
PK_ERROR_bad_light_source	364
PK_ERROR_eye_in_box	367
PK_ERROR_cyclic_assy	503
PK_ERROR_anon_sub_part	504

PK_ERROR_different_types	505
PK_ERROR_existing_attdef	506
PK_ERROR_R1_R2_mismatch	507
PK_ERROR_radius_sum_le_0	508
PK_ERROR_wrong_list_type	509
PK_ERROR_bad_tag_in_list	510
PK_ERROR_duplicate_array_item	511
PK_ERROR_not_in_group	512
PK_ERROR_wrong_class_for_group	513
PK_ERROR_array_too_short	519
PK_ERROR_already_in_group	520
PK_ERROR_attr_mismatch	522
PK_ERROR_list_wrong_length	523
PK_ERROR_part_not_keyed	524
PK_ERROR_cant_heal_wound	525
PK_ERROR_already_loaded	528
PK_ERROR_already_saved	529
PK_ERROR_key_in_use	530
PK_ERROR_closed_faces	531
PK_ERROR_at_singularity	532
PK_ERROR_size_mismatch	533
PK_ERROR_duplicate_tools	540
PK_ERROR_instanced_tools	541
PK_ERROR_mixed_sheets_solids	542
PK_ERROR_cant_unite_solid_sheet	543
PK_ERROR_same_tool_and_target	545
PK_ERROR_invalid_bodies	546
PK_ERROR_non_manifold	547
PK_ERROR_t_sheet	549
PK_ERROR_wrong_sub_type	553
PK_ERROR_attr_defn_mismatch	555
PK_ERROR_cant_find_file	557
PK_ERROR_get_snapshot_failed	558

PK_ERROR_transmit_failed	560
PK_ERROR_bad_filename	561
PK_ERROR_save_snapshot_failed	562
PK_ERROR_bad_key	565
PK_ERROR_journal_not_open	566
PK_ERROR_bad_state_combn	570
PK_ERROR_rollmark_failed	850
PK_ERROR_no_rollmark	854
PK_ERROR_roll_is_off	855
PK_ERROR_roll_forward_fail	856
PK_ERROR_impossible_taper	860
PK_ERROR_system_error	900
PK_ERROR_memory_full	901
PK_ERROR_nitems_lt_0	902
PK_ERROR_nitems_le_0	903
PK_ERROR_modified_sub_part	904
PK_ERROR_part_not_isolated	905
PK_ERROR_null_arg_address	906
PK_ERROR_bad_option_data	907
PK_ERROR_not_a_logical	908
PK_ERROR_bad_box	909
PK_ERROR_bad_position	911
PK_ERROR_empty_assy	912
PK_ERROR_keyed_part_mismatch	913
PK_ERROR_unsuitable_entity	914
PK_ERROR_not_on_surface	915
PK_ERROR_bad_shared_entity	916
PK_ERROR_bad_shared_dep	917
PK_ERROR_attr_type_not_defined	919
PK_ERROR_bad_blend_param	920
PK_ERROR_bad_sharing	921
PK_ERROR_corrupt_file	922
PK_ERROR_wrong_version	923

PK_ERROR_not_at_rollmark	924
PK_ERROR_radius_eq_0	925
PK_ERROR_radius_too_large	926
PK_ERROR_distance_too_large	927
PK_ERROR_cant_open_file	928
PK_ERROR_at_terminator	929
PK_ERROR_bad_precision	930
PK_ERROR_modeller_not_started	931
PK_ERROR_modeller_not_stopped	932
PK_ERROR_bad_user_field_size	933
PK_ERROR_recursive_call	934
PK_ERROR_bad_hull	935
PK_ERROR_usfd_mismatch	936
PK_ERROR_wrong_format	937
PK_ERROR_wire_body	938
PK_ERROR_not_sheet	939
PK_ERROR_bad_wire	940
PK_ERROR_bad_end_points	941
PK_ERROR_crossing_edge	942
PK_ERROR_crossing_vertex	943
PK_ERROR_bad_vertex	944
PK_ERROR_aborted	945
PK_ERROR_not_interrupted	946
PK_ERROR_run_time_error	947
PK_ERROR_fatal_error	948
PK_ERROR_no_user_fields	949
PK_ERROR_wrong_surface	950
PK_ERROR_opposed_sheets	951
PK_ERROR_coplanar	952
PK_ERROR_bad_accuracy	956
PK_ERROR_coincident	957
PK_ERROR_atol_too_small	958
PK_ERROR_ctol_too_small	959

PK_ERROR_stol_too_small	960
PK_ERROR_wrong_direction	961
PK_ERROR_non_orth_matrix	962
PK_ERROR_bad_component	963
PK_ERROR_bad_rollfile_size	964
PK_ERROR_cant_be_aborted	965
PK_ERROR_hulls_intersect	966
PK_ERROR_abort_from_go	967
PK_ERROR_all_faces_in_body	969
PK_ERROR_schema_access_error	970
PK_ERROR_schema_corrupt	971
PK_ERROR_cant_intsc_solid_sheet	972
PK_ERROR_file_access_error	973
PK_ERROR_bad_file_format	974
PK_ERROR_bad_file_guise	975
PK_ERROR_bad_rolling_ball	976
PK_ERROR_coincident_points	977
PK_ERROR_bad_knots	978
PK_ERROR_bad_derivative	979
PK_ERROR_wrong_number_knots	980
PK_ERROR_wrong_number_derivs	981
PK_ERROR_incompatible_props	982
PK_ERROR_repeated_knots	983
PK_ERROR_curves_dont_meet	984
PK_ERROR_insufficient_curves	985
PK_ERROR_bad_curves	986
PK_ERROR_bad_order	987
PK_ERROR_insufficient_points	988
PK_ERROR_bad_parametric_prop	989
PK_ERROR_illegal_owner	990
PK_ERROR_unchecked_entity	991
PK_ERROR_incompatible_curves	992
PK_ERROR_cant_make_bspline	993

PK_ERROR_cu_are_coincident	994
PK_ERROR_withdrawn_surface	995
PK_ERROR_face_not_planar	996
PK_ERROR_request_not_supported	997
PK_ERROR_contradictory_request	998
PK_ERROR_invalid_geometry	999

C.1.3 Error Codes 1000-4999

PK_ERROR_file_already_exists	1000
PK_ERROR_too_many_control_pts	1001
PK_ERROR_bad_string	1002
PK_ERROR_mend_attempt_failure	1003
PK_ERROR_bad_tag_in_list_tree	1004
PK_ERROR_bad_list_tree	1005
PK_ERROR_cyclic_list_reference	1006
PK_ERROR_empty_list_in_tree	1007
PK_ERROR_cant_make_trimmed_sf	1008
PK_ERROR_bad_entity_event_comb	1009
PK_ERROR_too_many_derivatives	1010
PK_ERROR_bad_deriv_vertices	1011
PK_ERROR_bad_degen_vertices	1012
PK_ERROR_not_on_edge	1013
PK_ERROR_closest_approach_failed	1014
PK_ERROR_cant_do_clash	1015
PK_ERROR_targ_faces_many_bodies	1016
PK_ERROR_tool_faces_many_bodies	1017
PK_ERROR_cant_do_imprint	1018
PK_ERROR_topol_not_from_body	1019
PK_ERROR_inconsistent_facesets	1020
PK_ERROR_FG_evaluator_not_found	1021
PK_ERROR_FG_data_alloc_error	1022
PK_ERROR_FG_data_not_found	1023

PK_ERROR_FG_evaluator_error	1024
PK_ERROR_FG_modelling_error	1025
PK_ERROR_solid_body	1026
PK_ERROR_different_bodies	1027
PK_ERROR_wrong_number_edges	1028
PK_ERROR_cant_blend_vertex	1029
PK_ERROR_blends_overlap	1030
PK_ERROR_edges_intersect	1031
PK_ERROR_not_in_same_body	1032
PK_ERROR_unsuitable_topology	1033
PK_ERROR_cu_self_intersect	1034
PK_ERROR_linear_multi_seg	1035
PK_ERROR_no_eds_from_target	1036
PK_ERROR_cant_offset	1037
PK_ERROR_FG_real_data_error	1038
PK_ERROR_FG_integer_data_error	1039
PK_ERROR_partial_coi_found	1040
PK_ERROR_bodies_dont_knit	1041
PK_ERROR_pattern_invalid	1042
PK_ERROR_bad_tolerance	1043
PK_ERROR_cant_extract_geom	1044
PK_ERROR_bad_basis_surf	1045
PK_ERROR_FG_receive_failure	1046
PK_ERROR_FG_snapshot_failure	1047
PK_ERROR_cant_create_pattern	1048
PK_ERROR_tag_limit_exceeded	1049
PK_ERROR_tag_limit_out_of_range	1050
PK_ERROR_cant_find_extreme	1051
PK_ERROR_disc_full	1052
PK_ERROR_cant_find_derivs	1053
PK_ERROR_too_many_targets	1054
PK_ERROR_duplicate_targets	1055
PK_ERROR_curve_already_trimmed	1056

PK_ERROR_curve_too_short	1057
PK_ERROR_boolean_failure	1058
PK_ERROR_duplicate_item	1059
PK_ERROR_failed_to_trim	1060
PK_ERROR_unsuitable_loop	1061
PK_ERROR_failed_to_replace	1062
PK_ERROR_failed_to_create_sp	1063
PK_ERROR_tolerances_too_tight	1064
PK_ERROR_fru_error	1065
PK_ERROR_incorrect_mc_conf	1066
PK_ERROR_partial_no_intersect	1067
PK_ERROR_none_shared	1068
PK_ERROR_cant_hollow	1069
PK_ERROR_not_in_same_shell	1070
PK_ERROR_general_body	1071
PK_ERROR_bad_thickness	1072
PK_ERROR_non_smooth_edge	1073
PK_ERROR_degenerate_vertex	1074
PK_ERROR_cant_thicken	1075
PK_ERROR_crossing_face	1076
PK_ERROR_not_in_region	1077
PK_ERROR_empty_body	1078
PK_ERROR_sheet_untrimmed	1079
PK_ERROR_fxf_blend_failed	1080
PK_ERROR_fxf_blend_bad_token	1081
PK_ERROR_file_read_corruption	1082
PK_ERROR_trim_loop_degenerate	1083
PK_ERROR_solid_has_void	1084
PK_ERROR_fru_missing	1085
PK_ERROR_not_in_same_partition	1086
PK_ERROR_instanced_body	1087
PK_ERROR_entity_not_new	1088
PK_ERROR_applio_not_registered	1089

PK_ERROR_more_than_one_part	1090
PK_ERROR_bad_field_conversion	1091
PK_ERROR_bad_text_conversion	1092

C.1.4 Error Codes 5000-9999

PK_ERROR_not_implemented	5000
PK_ERROR_not_in_PK	5001
PK_ERROR_unknown_class	5002
PK_ERROR_frustrum_failure	5003
PK_ERROR_recursion_depth	5004
PK_ERROR_not_a_part	5005
PK_ERROR_bad_mark	5006
PK_ERROR_mark_ki_started	5007
PK_ERROR_mark_not_started	5008
PK_ERROR_num_derivs_not_equal	5009
PK_ERROR_memory_not_empty	5010
PK_ERROR_no_last_error	5011
PK_ERROR_zero_bytes_required	5012
PK_ERROR_bad_field_number	5013
PK_ERROR_field_of_wrong_type	5014
PK_ERROR_invalid_group_class	5015
PK_ERROR_unhandleable_condition	5016
PK_ERROR_wrong_group_class	5017
PK_ERROR_eval_failure	5018
PK_ERROR_not_a_unit_vector	5019
PK_ERROR_existing_attrib	5020
PK_ERROR_solid_region	5021
PK_ERROR_o_t_version_unknown	5022
PK_ERROR_mark_pk_started	5023
PK_ERROR_vectors_are_parallel	5024
PK_ERROR_sum_of_offsets_eq_0	5025
PK_ERROR_distance_too_small	5026

PK_ERROR_vectors_not_orthogonal	5027
PK_ERROR_zero_interval	5028
PK_ERROR_periodic_open	5029
PK_ERROR_periodic_not_smooth	5030
PK_ERROR_cant_get_point	5031
PK_ERROR_cant_get_curve	5032
PK_ERROR_edge_didnt_vanish	5033
PK_ERROR_face_face_check_fails	5034
PK_ERROR_face_check_fails	5035
PK_ERROR_not_a_hand	5036
PK_ERROR_bad_interval	5037
PK_ERROR_bad_uvbox	5038
PK_ERROR_cannot_make_current	5039
PK_ERROR_bb_not_empty	5040
PK_ERROR_bad_class	5041
PK_ERROR_bad_class_combn	5042
PK_ERROR_o_t_version_incorrect	5043
PK_ERROR_bad_boolean_function	5044
PK_ERROR_bad_boolean_region	5045
PK_ERROR_distancing_failed	5046
PK_ERROR_no_overlap	5047
PK_ERROR_rollback_not_started	5048
PK_ERROR_rollback_started	5049
PK_ERROR_cant_get_surf	5050
PK_ERROR_bad_section_fence	5051
PK_ERROR_bad_boolean_fence	5052
PK_ERROR_cant_get_side_curve	5053
PK_ERROR_cant_get_side_surf	5054
PK_ERROR_sewing_failed	5055
PK_ERROR_duplicate_parts	5056
PK_ERROR_deltas_not_available	5057
PK_ERROR_not_at_pmark	5058
PK_ERROR_bad_boolean_match	5059

PK_ERROR_invalid_match_region	5060
PK_ERROR_check_error	5061
PK_ERROR_check_failure	5062
PK_ERROR_no_approx_data	5063
PK_ERROR_not_general	5064
PK_ERROR_cant_merge_regions	5065
PK_ERROR_not_a_bb	5066
PK_ERROR_bad_bb_status	5067
PK_ERROR_unsupported_operation	5068
PK_ERROR_bad_end_condition	5069
PK_ERROR_bad_boolean_select	5070
PK_ERROR_bad_thread	5071
PK_ERROR_no_intersect	5072
PK_ERROR_obsolete_function	5073
PK_ERROR_reverse_edge_failed	5074
PK_ERROR_orientation_failed	5075
PK_ERROR_bad_edge	5076
PK_ERROR_curve_nmnl_off	5077
PK_ERROR_bad_pattern_check	5078
PK_ERROR_bad_pattern_status	5079
PK_ERROR_bad_pattern_result	5080
PK_ERROR_wrong_diversion_delta	5081
PK_ERROR_edge_not_open	5082
PK_ERROR_edge_not_manifold	5083
PK_ERROR_vertex_not_manifold	5084
PK_ERROR_no_common_vertex	5085
PK_ERROR_fin_not_at_vertex	5086
PK_ERROR_bad_fin	5087
PK_ERROR_not_laminar_side	5088
PK_ERROR_wireframe_edge	5089
PK_ERROR_different_shells	5090
PK_ERROR_face_contains_loop	5091
PK_ERROR_only_one_loop	5092

PK_ERROR_invalid_face	5093
PK_ERROR_fins_not_distinct	5094
PK_ERROR_bad_check_topol	5095
PK_ERROR_bad_check_su_X	5096
PK_ERROR_bad_change_topol	5097
PK_ERROR_bad_local_status	5098
PK_ERROR_bad_topol_track	5099
PK_ERROR_fin_not_in_loop	5100
PK_ERROR_too_many_edges	5101
PK_ERROR_loop_not_isolated	5102
PK_ERROR_loop_in_different_faces	5103
PK_ERROR_bad_boolean_report	5104
PK_ERROR_bad_boolean_result	5105
PK_ERROR_bad_section_report	5106
PK_ERROR_bad_section_result	5107
PK_ERROR_edge_is_wire	5108
PK_ERROR_bad_boolean_check_fa	5109
PK_ERROR_bad_section_check_fa	5110
PK_ERROR_inconsistent_offset	5111
PK_ERROR_inconsistent_thicken	5112
PK_ERROR_bad_twist_law	5113
PK_ERROR_bad_scale_law	5114
PK_ERROR_bad_profile_matching	5115
PK_ERROR_bad_end_conditions	5116
PK_ERROR_loft_failed	5117
PK_ERROR_sweep_failed	5118
PK_ERROR_different_loops	5119
PK_ERROR_edge_is_ring	5120
PK_ERROR_loops_not_distant	5121
PK_ERROR_point_contact	5122
PK_ERROR_sweep_need_piecewise	5123
PK_ERROR_zero_vector	5124
PK_ERROR_bad_iteration_count	5125

PK_ERROR_bad_gap_bound	5126
PK_ERROR_bad_match_style	5127
PK_ERROR_no_division	5128
PK_ERROR_no_nth_division	5129
PK_ERROR_bad_reference	5130
PK_ERROR_cant_fill_hole	5131
PK_ERROR_no_data	5132
PK_ERROR_too_many_geoms	5133
PK_ERROR_disjoint	5134
PK_ERROR_duplicate_name	5135
PK_ERROR_failed_to_make_outline	5136
PK_ERROR_failed_to_blend	5137
PK_ERROR_failed_to_offset	5138
PK_ERROR_failed_to_taper	5139
PK_ERROR_failed_to_transform	5140
PK_ERROR_journalling_on	5141
PK_ERROR_dbg_rprt_not_stopped	5142
PK_ERROR_dbg_rprt_not_started	5143
PK_ERROR_invalid_object	9999

PK Error Codes (Alphabetic)

D

D.1 Alphabetical List of PK Error Codes

This appendix lists all the error codes used by the PK Interface in alphabetical order.

D.1.1 Error Codes FG*

PK_ERROR_FG_data_alloc_error	1022
PK_ERROR_FG_data_not_found	1023
PK_ERROR_FG_evaluator_error	1024
PK_ERROR_FG_evaluator_not_found	1021
PK_ERROR_FG_integer_data_error	1039
PK_ERROR_FG_modelling_error	1025
PK_ERROR_FG_real_data_error	1038
PK_ERROR_FG_receive_failure	1046
PK_ERROR_FG_snapshot_failure	1047

D.1.2 Error Codes a-b

PK_ERROR_abort_from_go	967
PK_ERROR_aborted	945
PK_ERROR_all_faces_in_body	969
PK_ERROR_already_in_group	520
PK_ERROR_already_loaded	528
PK_ERROR_already_saved	529
PK_ERROR_anon_sub_part	504
PK_ERROR_applio_not_registered	1089
PK_ERROR_array_too_short	519
PK_ERROR_at_singularity	532
PK_ERROR_at_terminator	929

PK_ERROR_atol_too_small	958
PK_ERROR_attr_defn_mismatch	555
PK_ERROR_attr_mismatch	522
PK_ERROR_attr_not_found	99
PK_ERROR_attr_type_not_defined	919
PK_ERROR_bad_accuracy	956
PK_ERROR_bad_angle	1
PK_ERROR_bad_bb_status	5067
PK_ERROR_bad_basis_surf	1045
PK_ERROR_bad_blend_bound	334
PK_ERROR_bad_blend_param	920
PK_ERROR_bad_boolean_check_fa	5109
PK_ERROR_bad_boolean_fence	5052
PK_ERROR_bad_boolean_function	5044
PK_ERROR_bad_boolean_match	5059
PK_ERROR_bad_boolean_region	5045
PK_ERROR_bad_boolean_report	5104
PK_ERROR_bad_boolean_result	5105
PK_ERROR_bad_boolean_select	5070
PK_ERROR_bad_box	909
PK_ERROR_bad_change_topol	5097
PK_ERROR_bad_char_string	109
PK_ERROR_bad_check_su_X	5096
PK_ERROR_bad_check_topol	5095
PK_ERROR_bad_class	5041
PK_ERROR_bad_class_combn	5042
PK_ERROR_bad_component	963
PK_ERROR_bad_curves	986
PK_ERROR_bad_degen_vertices	1012
PK_ERROR_bad_derivative	979
PK_ERROR_bad_deriv_vertices	1011
PK_ERROR_bad_dimension	135
PK_ERROR_bad_edge	5076

PK_ERROR_bad_end_condition	5069
PK_ERROR_bad_end_conditions	5116
PK_ERROR_bad_end_points	941
PK_ERROR_bad_entity_event_comb	1009
PK_ERROR_bad_field_conversion	1091
PK_ERROR_bad_field_number	5013
PK_ERROR_bad_file_format	974
PK_ERROR_bad_file_guise	975
PK_ERROR_bad_filename	561
PK_ERROR_bad_fin	5087
PK_ERROR_bad_gap_bound	5126
PK_ERROR_bad_geom_topol	105
PK_ERROR_bad_hull	935
PK_ERROR_bad_index	29
PK_ERROR_bad_interval	5037
PK_ERROR_bad_iteration_count	5125
PK_ERROR_bad_key	565
PK_ERROR_bad_knots	978
PK_ERROR_bad_light_source	364
PK_ERROR_bad_list_tree	1005
PK_ERROR_bad_local_status	5098
PK_ERROR_bad_mark	5006
PK_ERROR_bad_match_style	5127
PK_ERROR_bad_name	11
PK_ERROR_bad_option_data	907
PK_ERROR_bad_order	987
PK_ERROR_bad_parameter	120
PK_ERROR_bad_parametric_prop	989
PK_ERROR_bad_pattern_check	5078
PK_ERROR_bad_pattern_status	5079
PK_ERROR_bad_pattern_result	5080
PK_ERROR_bad_pixel_map	361
PK_ERROR_bad_position	911

PK_ERROR_bad_precision	930
PK_ERROR_bad_profile_matching	5115
PK_ERROR_bad_reference	5130
PK_ERROR_bad_request_code	350
PK_ERROR_bad_rollfile_size	964
PK_ERROR_bad_rolling_ball	976
PK_ERROR_bad_scale_law	5114
PK_ERROR_bad_section_check_fa	5110
PK_ERROR_bad_section_fence	5051
PK_ERROR_bad_section_report	5106
PK_ERROR_bad_section_result	5107
PK_ERROR_bad_selection_code	38
PK_ERROR_bad_shared_dep	917
PK_ERROR_bad_shared_entity	916
PK_ERROR_bad_sharing	921
PK_ERROR_bad_spec_code	110
PK_ERROR_bad_state_combn	570
PK_ERROR_bad_string	1002
PK_ERROR_bad_tag_in_list	510
PK_ERROR_bad_tag_in_list_tree	1004
PK_ERROR_bad_text_conversion	1092
PK_ERROR_bad_thickness	1072
PK_ERROR_bad_thread	5071
PK_ERROR_bad_tolerance	1043
PK_ERROR_bad_topol_track	5099
PK_ERROR_bad_twist_law	5113
PK_ERROR_bad_type	30
PK_ERROR_bad_type_combn	12
PK_ERROR_unsupported_operation	5068
PK_ERROR_bad_user_field_size	933
PK_ERROR_bad_uvbox	5038
PK_ERROR_bad_value	39
PK_ERROR_bad_vertex	944

PK_ERROR_bad_view_mx	360
PK_ERROR_bad_wire	940
PK_ERROR_bb_not_empty	5040
PK_ERROR_blend_didnt_check	336
PK_ERROR_blends_overlap	1030
PK_ERROR_bodies_dont_knit	1041
PK_ERROR_boolean_failure	1058
PK_ERROR_buffer_overflow	2
PK_ERROR_bulletinb_is_off	42

D.1.3 Error Codes c-f

PK_ERROR_cannot_make_current	5039
PK_ERROR_cant_be_aborted	965
PK_ERROR_cant_blend_vertex	1029
PK_ERROR_cant_create_pattern	1048
PK_ERROR_cant_do_clash	1015
PK_ERROR_cant_do_imprint	1018
PK_ERROR_cant_do_intersect	157
PK_ERROR_cant_do_tweak	50
PK_ERROR_cant_extract_geom	1044
PK_ERROR_cant_fill_hole	5131
PK_ERROR_cant_find_derivs	1053
PK_ERROR_cant_find_extreme	1051
PK_ERROR_cant_find_file	557
PK_ERROR_cant_find_su	77
PK_ERROR_cant_fix_blends	330
PK_ERROR_cant_get_curve	5032
PK_ERROR_cant_get_side_curve	5053
PK_ERROR_cant_get_side_surf	5054
PK_ERROR_cant_get_surf	5050
PK_ERROR_cant_get_point	5031
PK_ERROR_cant_heal_wound	525

PK_ERROR_cant_hollow	1069
PK_ERROR_cant_intsc_solid_sheet	972
PK_ERROR_cant_make_bspline	993
PK_ERROR_cant_make_trimmed_sf	1008
PK_ERROR_cant_merge_regions	5065
PK_ERROR_cant_offset	1037
PK_ERROR_cant_open_file	928
PK_ERROR_cant_open_jrnl	27
PK_ERROR_cant_thicken	1075
PK_ERROR_cant_unite_solid_sheet	543
PK_ERROR_check_error	5061
PK_ERROR_check_failure	5062
PK_ERROR_closed_faces	531
PK_ERROR_closest_approach_failed	1014
PK_ERROR_coincident	957
PK_ERROR_coincident_points	977
PK_ERROR_cone_too_sharp	4
PK_ERROR_contradictory_request	998
PK_ERROR_coplanar	952
PK_ERROR_corrupt_body	103
PK_ERROR_corrupt_file	922
PK_ERROR_crossing_edge	942
PK_ERROR_crossing_face	1076
PK_ERROR_crossing_vertex	943
PK_ERROR_ctol_too_small	959
PK_ERROR_cu_are_coincident	994
PK_ERROR_cu_self_intersect	1034
PK_ERROR_curve_already_trimmed	1056
PK_ERROR_curve_nmnl_off	5077
PK_ERROR_curve_too_short	1057
PK_ERROR_curves_dont_meet	984
PK_ERROR_cyclic_assy	503
PK_ERROR_cyclic_list_reference	1006

PK_ERROR_dbg_rprt_not_started	5143
PK_ERROR_dbg_rprt_not_stopped	5142
PK_ERROR_degenerate_vertex	1074
PK_ERROR_deltas_not_available	5057
PK_ERROR_density_le_0	85
PK_ERROR_different_bodies	1027
PK_ERROR_different_loops	5119
PK_ERROR_different_shells	5090
PK_ERROR_different_types	505
PK_ERROR_disc_full	1052
PK_ERROR_discontinuous_curve	131
PK_ERROR_discontinuous_surface	129
PK_ERROR_disjoint	5134
PK_ERROR_distance_le_0	15
PK_ERROR_distance_lt_0	14
PK_ERROR_distance_too_small	5026
PK_ERROR_distance_too_large	927
PK_ERROR_distancing_failed	5046
PK_ERROR_dont_intersect	35
PK_ERROR_dont_make_solid	87
PK_ERROR_duplicate_array_item	511
PK_ERROR_duplicate_item	1059
PK_ERROR_duplicate_name	5135
PK_ERROR_duplicate_parts	5056
PK_ERROR_duplicate_targets	1055
PK_ERROR_duplicate_tools	540
PK_ERROR_edge_didnt_vanish	5033
PK_ERROR_edge_is_ring	5120
PK_ERROR_edge_is_wire	5108
PK_ERROR_edge_not_manifold	5083
PK_ERROR_edge_not_open	5082
PK_ERROR_edges_intersect	1031
PK_ERROR_empty_assy	912

PK_ERROR_empty_body	1078
PK_ERROR_empty_list	79
PK_ERROR_empty_list_in_tree	1007
PK_ERROR_entity_not_new	1088
PK_ERROR_eval_failure	5018
PK_ERROR_existing_attdef	506
PK_ERROR_existing_attrib	5020
PK_ERROR_eye_in_box	367
PK_ERROR_face_check_fails	5035
PK_ERROR_face_contains_loop	5091
PK_ERROR_face_face_check_fails	5034
PK_ERROR_face_not_planar	996
PK_ERROR_failed_to_blend	5137
PK_ERROR_failed_to_create_sp	1063
PK_ERROR_failed_to_make_outline	5136
PK_ERROR_failed_to_offset	5138
PK_ERROR_failed_to_replace	1062
PK_ERROR_failed_to_taper	5139
PK_ERROR_failed_to_transform	5140
PK_ERROR_failed_to_trim	1060
PK_ERROR_fatal_error	948
PK_ERROR_field_of_wrong_type	5014
PK_ERROR_file_access_error	973
PK_ERROR_file_already_exists	1000
PK_ERROR_file_read_corruption	1082
PK_ERROR_fin_not_in_loop	5100
PK_ERROR_fin_not_at_vertex	5086
PK_ERROR_fins_not_distinct	5094
PK_ERROR_fragment	73
PK_ERROR_fru_error	1065
PK_ERROR_fru_missing	1085
PK_ERROR_frustrum_failure	5003

PK_ERROR_fxf_blend_failed	1080
PK_ERROR_fxf_blend_bad_token	1081

D.1.4 Error Codes g-o

PK_ERROR_general_body	1071
PK_ERROR_geom_not_needed	64
PK_ERROR_geom_topol_mismatch	62
PK_ERROR_get_snapshot_failed	558
PK_ERROR_has_no_name	7
PK_ERROR_has_no_owner	8
PK_ERROR_has_parent	28
PK_ERROR_hulls_intersect	966
PK_ERROR_illegal_degeneracy	116
PK_ERROR_illegal_owner	990
PK_ERROR_impossible_sweep	57
PK_ERROR_impossible_swing	55
PK_ERROR_impossible_taper	860
PK_ERROR_incompatible_curves	992
PK_ERROR_incompatible_props	982
PK_ERROR_inconsistent_facesets	1020
PK_ERROR_inconsistent_geom	51
PK_ERROR_inconsistent_offset	5111
PK_ERROR_inconsistent_thicken	5112
PK_ERROR_incorrect_mc_conf	1066
PK_ERROR_instanced_body	1087
PK_ERROR_instanced_tools	541
PK_ERROR_insufficient_curves	985
PK_ERROR_insufficient_points	988
PK_ERROR_invalid_bodies	546
PK_ERROR_invalid_face	5093
PK_ERROR_invalid_geometry	999
PK_ERROR_invalid_group_class	5015

PK_ERROR_invalid_match_region	5060
PK_ERROR_invalid_object	9999
PK_ERROR_is_attached	34
PK_ERROR_journal_not_open	566
PK_ERROR_journalling_on	5141
PK_ERROR_key_in_use	530
PK_ERROR_key_not_found	58
PK_ERROR_keyed_part_mismatch	913
PK_ERROR_linear_multi_seg	1035
PK_ERROR_list_wrong_length	523
PK_ERROR_loft_failed	5117
PK_ERROR_loop_in_different_faces	5103
PK_ERROR_loop_not_isolated	5102
PK_ERROR_loops_not_distant	5121
PK_ERROR_lt_3_sides	33
PK_ERROR_majaxi_not_perpn	36
PK_ERROR_mark_ki_started	5007
PK_ERROR_mark_not_started	5008
PK_ERROR_mark_pk_started	5023
PK_ERROR_mass_eq_0	82
PK_ERROR_memory_full	901
PK_ERROR_memory_not_empty	5010
PK_ERROR_mend_attempt_failure	1003
PK_ERROR_missing_geom	96
PK_ERROR_mixed_sheets_solids	542
PK_ERROR_modeller_not_started	931
PK_ERROR_modeller_not_stopped	932
PK_ERROR_modified_sub_part	904
PK_ERROR_more_than_one_part	1090
PK_ERROR_negative_body	106
PK_ERROR_nitems_le_0	903
PK_ERROR_nitems_lt_0	902
PK_ERROR_no_approx_data	5063

PK_ERROR_no_common_vertex	5085
PK_ERROR_no_data	5132
PK_ERROR_no_division	5128
PK_ERROR_no_eds_from_target	1036
PK_ERROR_no_errors	0
PK_ERROR_no_geometry	61
PK_ERROR_no_last_error	5011
PK_ERROR_no_intersect	5072
PK_ERROR_no_nth_division	5129
PK_ERROR_no_overlap	5047
PK_ERROR_no_rollmark	854
PK_ERROR_no_user_fields	949
PK_ERROR_non_manifold	547
PK_ERROR_non_orth_matrix	962
PK_ERROR_non_smooth_edge	1073
PK_ERROR_none_mergeable	48
PK_ERROR_none_shared	1068
PK_ERROR_not_a_bb	5066
PK_ERROR_not_a_hand	5036
PK_ERROR_not_a_list	80
PK_ERROR_not_a_logical	908
PK_ERROR_not_a_part	5005
PK_ERROR_not_a_unit_vector	5019
PK_ERROR_not_general	5064
PK_ERROR_not_implemented	5000
PK_ERROR_not_in_PK	5001
PK_ERROR_not_an_entity	22
PK_ERROR_not_at_pmark	5058
PK_ERROR_not_at_rollmark	924
PK_ERROR_not_blended	335
PK_ERROR_not_connected	20
PK_ERROR_not_found	19
PK_ERROR_not_in_group	512

PK_ERROR_not_in_region	1077
PK_ERROR_not_in_same_body	1032
PK_ERROR_not_in_same_part	59
PK_ERROR_not_in_same_partition	1086
PK_ERROR_not_in_same_shell	1070
PK_ERROR_not_interrupted	946
PK_ERROR_not_laminar_side	5088
PK_ERROR_not_on_curve	67
PK_ERROR_not_on_edge	1013
PK_ERROR_not_on_face	54
PK_ERROR_not_on_surface	915
PK_ERROR_not_same_length	359
PK_ERROR_not_sheet	939
PK_ERROR_not_solid	101
PK_ERROR_not_unique	13
PK_ERROR_null_arg_address	906
PK_ERROR_null_axis	25
PK_ERROR_null_direction	31
PK_ERROR_num_derivs_not_equal	5009
PK_ERROR_o_t_version_incorrect	5043
PK_ERROR_o_t_version_unknown	5022
PK_ERROR_obsolete_function	5073
PK_ERROR_only_one_loop	5092
PK_ERROR_opposed_sheets	951
PK_ERROR_order_lt_2	132
PK_ERROR_orientation_failed	5075

D.1.5 Error Codes p-z

PK_ERROR_part_not_isolated	905
PK_ERROR_part_not_keyed	524
PK_ERROR_partial_coi_found	1040
PK_ERROR_partial_no_intersect	1067

PK_ERROR_pattern_invalid	1042
PK_ERROR_periodic_not_smooth	5030
PK_ERROR_periodic_open	5029
PK_ERROR_point_contact	5122
PK_ERROR_R1_R2_mismatch	507
PK_ERROR_radii_both_0	3
PK_ERROR_radius_eq_0	925
PK_ERROR_radius_le_0	16
PK_ERROR_radius_lt_0	18
PK_ERROR_radius_sum_le_0	508
PK_ERROR_radius_too_large	926
PK_ERROR_receive_failed	63
PK_ERROR_recursion_depth	5004
PK_ERROR_recursive_call	934
PK_ERROR_repeated_knots	983
PK_ERROR_request_not_supported	997
PK_ERROR_reverse_edge_failed	5074
PK_ERROR_roll_forward_fail	856
PK_ERROR_roll_is_off	855
PK_ERROR_rollback_not_started	5048
PK_ERROR_rollback_started	5049
PK_ERROR_rollmark_failed	850
PK_ERROR_rot_angle_eq_0	32
PK_ERROR_run_time_error	947
PK_ERROR_same_tool_and_target	545
PK_ERROR_save_snapshot_failed	562
PK_ERROR_sc_factor_le_0	40
PK_ERROR_schema_access_error	970
PK_ERROR_schema_corrupt	971
PK_ERROR_sewing_failed	5055
PK_ERROR_sheet_untrimmed	1079
PK_ERROR_size_mismatch	533
PK_ERROR_solid_body	1026

PK_ERROR_solid_has_void	1084
PK_ERROR_solid_region	5021
PK_ERROR_still_referenced	68
PK_ERROR_stol_too_small	960
PK_ERROR_su_are_coincident	41
PK_ERROR_su_self_intersect	141
PK_ERROR_sum_of_offsets_eq_0	5025
PK_ERROR_sweep_failed	5118
PK_ERROR_sweep_need_piecewise	5123
PK_ERROR_system_error	900
PK_ERROR_t_sheet	549
PK_ERROR_tag_limit_exceeded	1049
PK_ERROR_tag_limit_out_of_range	1050
PK_ERROR_targ_faces_many_bodies	1016
PK_ERROR_tolerances_too_tight	1064
PK_ERROR_too_many_control_pts	1001
PK_ERROR_too_many_derivatives	1010
PK_ERROR_too_many_edges	5101
PK_ERROR_too_many_geoms	5133
PK_ERROR_too_many_targets	1054
PK_ERROR_tool_faces_many_bodies	1017
PK_ERROR_topol_not_from_body	1019
PK_ERROR_transmit_failed	560
PK_ERROR_trim_loop_degenerate	1083
PK_ERROR_unchecked_entity	991
PK_ERROR_unhandleable_condition	5016
PK_ERROR_unknown_class	5002
PK_ERROR_unsuitable_entity	914
PK_ERROR_unsuitable_loop	1061
PK_ERROR_unsuitable_topology	1033
PK_ERROR_usfd_mismatch	936
PK_ERROR_vectors_are_parallel	5024
PK_ERROR_vectors_not_orthogonal	5027

PK_ERROR_vertex_not_manifold	5084
PK_ERROR_weight_le_0	111
PK_ERROR_wire_body	938
PK_ERROR_wireframe_edge	5089
PK_ERROR_withdrawn_surface	995
PK_ERROR_wrong_class_for_group	513
PK_ERROR_wrong_direction	961
PK_ERROR_wrong_diversion_delta	5081
PK_ERROR_wrong_entity	10
PK_ERROR_wrong_entity_in_array	357
PK_ERROR_wrong_format	937
PK_ERROR_wrong_group_class	5017
PK_ERROR_wrong_list_type	509
PK_ERROR_wrong_number_derivs	981
PK_ERROR_wrong_number_edges	1028
PK_ERROR_wrong_number_knots	980
PK_ERROR_wrong_sub_type	553
PK_ERROR_wrong_surface	950
PK_ERROR_wrong_transf	37
PK_ERROR_wrong_version	923
PK_ERROR_zero_bytes_required	5012
PK_ERROR_zero_interval	5028
PK_ERROR_zero_vector	5124