uc5000 (view source)

Defined in: uf_part.h

Overview
uc5000 check for active part - OLD NAMING CONVENTION

Required License(s)
gateway

int uc5000
(
char cr2 [MAX_FSPEC_BUFSIZE]
)

char	**cr2 [MAX_FSPEC_BUFSIZE]**	Output

---

uc5001 (view source)

Defined in: uf_part.h

Overview
uc5001 retrieve part file

Required License(s)
gateway

int uc5001
(
const char * cp1
)

const char *	**cp1**	Input

---

uc5003 (view source)

Defined in: uf_part.h

Overview
uc5003 file current part - OLD NAMING CONVENTION

Required License(s)
gateway

int uc5003
(

```
    const char * cp1,
    int ip2
)
```

const char *	**cp1**	Input	
int	**ip2**		

UF_PART_add_to_recent_file_list (view source)

Defined in: uf_part_ugopenint.h

Overview
Adds a part to the top of the "recently opened parts" list used by the NX user interface. If the part already exists on the "recently opened parts" list, it will be moved to the top of the list.

Environment
Internal

History
Originally released in V17

Required License(s)
gateway

```
int UF_PART_add_to_recent_file_list
(
    tag_t part_tag
)
```

tag_t	**part_tag**	Input	tag of part to add to list

UF_PART_apply_family_instance (view source)

Defined in: uf_part.h

Overview
Apply the attribute values of a member to the family part

Environment
Internal and External

Required License(s)
solid_modeling

```
int UF_PART_apply_family_instance
(
    tag_t family,
    int member_index
```

)

tag_t	**family**	Input
int	**member_index**	Input

UF_PART_ask_compression_flags (view source)

Defined in: uf_part.h

Overview
Reads the compression flags of the part file. If the standard field of the UF_PART_compress_flags_s structure is true, then when the part is saved, it is saved in compressed format.

Environment
Internal and External

See Also
UF_PART_set_compression_flags
UF_PART_compress_flags_p_t
Refer to the example

Required License(s)
gateway

int UF_PART_ask_compression_flags
(
tag_t part,
UF_PART_compress_flags_p_t compress_mask
)

tag_t	**part**	Input	Part to ask about compression flag
UF_PART_compress_flags_p_t	**compress_mask**	Output	bit field of compression flags settings

---

UF_PART_ask_customer_area (view source)

Defined in: uf_part.h

Overview
Returns a character pointer to an allocated copy of the customer area.

Environment
Internal and External

See Also
UF_PART_set_customer_area

Required License(s)
gateway

int UF_PART_ask_customer_area
(
tag_t part_tag,
char * * customer_area
)

tag_t	**part_tag**	Input	Tag of the part
char * *	**customer_area**	Output to UF_*free*	Pointer to the copy of the customer area. Use UF_free to deallocate memory. If there is no customer area, can potentially return a NULL pointer.

UF_PART_ask_description (view source)

Defined in: uf_part.h

Overview
Returns a character pointer to an allocated copy of the part description area.

Environment
Internal and External

See Also
UF_PART_set_description

Required License(s)
gateway

int UF_PART_ask_description
(
tag_t part_tag,
char * * description
)

tag_t	**part_tag**	Input	Tag of the part
char * *	**description**	Output to UF_*free*	Pointer to the copy of the part description area. Use UF_free to deallocate memory. If there is no description area, can potentially return a NULL pointer.

UF_PART_ask_display_part (view source)

Defined in: uf_part.h

Overview

Returns the tag of the current display part. In a non-assembly part, this is the same as the work part. If there currently isn't a displayed part, a NULL_TAG is returned.

Return
Tag of currently displayed part

Environment
Internal and External

Required License(s)
gateway

```
tag_t UF_PART_ask_display_part
(
    void
)
```

UF_PART_ask_enforce_piece_part (view source)

Defined in: uf_part.h

Overview
Asks the enforce piece part status of a part.

Environment
Internal and External

Required License(s)
gateway

```
int UF_PART_ask_enforce_piece_part
(
    tag_t part_tag,
    logical * status
)
```

tag_t	**part_tag**	Input	The tag of the part
logical *	**status**	Output	The enforce piece part status of the part. True, if part is an enforced piece part, false otherwise

---

UF_PART_ask_fam_inst_save_dir (view source)

Defined in: uf_part.h

Overview
Returns the current part family instance save directory.

Environment

Internal and External

Required License(s)
solid_modeling

int UF_PART_ask_fam_inst_save_dir
(
char * * part_directory
)

char * *	**part_directory**	Output to UF_*free*	Directory Specification, use UF_free to deallocate memory when done.

UF_PART_ask_families (view source)

Defined in: uf_part.h

Overview
Returns the part families in a part.

Environment
Internal and External

Required License(s)
gateway

int UF_PART_ask_families
(
tag_t part,
int * family_count,
tag_t * * families
)

tag_t	**part**	Input	Tag of the part. If you have a part occurrence, use UF_ASSEM_ask_prototype_of_occ to get the tag of the part to pass into this routine. If you have a part instance, use UF_ASSEM_ask_parent_of_instance to get the tag of the part to pass into this routine.
int *	**family_count**	Output	Number of part families in the part.
tag_t * *	**families**	Output to UF_*free*	Object identifiers of the part families. Use UF_free to deallocate memory when done.

UF_PART_ask_family_instance (view source)

Defined in: uf_part.h

Overview
Returns the object identifier of the part family instance within a part.

Environment
Internal and External

Required License(s)
gateway

int UF_PART_ask_family_instance
(
tag_t part,
tag_t * instance
)

tag_t	**part**	Input	Tag of the part. If you have a part occurrence, use UF_ASSEM_ask_prototype_of_occ to get the tag of the part to pass into this routine. If you have a part instance, use UF_ASSEM_ask_parent_of_instance to get the tag of the part to pass into this routine.
tag_t *	**instance**	Output	Object identifier of the part family instance.

UF_PART_ask_family_save_dir (view source)

Defined in: uf_part.h

Overview
Returns the directory in which members of the given family are saved.
If no directory was specified by a previous call to
UF_PART_set_family_save_dir then it defaults to the value returned
by UF_PART_set_fam_inst_save_dir.

Environment
Internal and External

See Also
UF_PART_set_family_save_dir

History
This function was originally released in V15.0.

Required License(s)
gateway

int UF_PART_ask_family_save_dir
(
tag_t family,
char * * dir
)

tag_t	**family**	Input	Tag of the family to ask the default save directory

char * *	**dir**	Output to UF_*free*	Directory. This must be freed by calling UF_free.

UF_PART_ask_jt_info_of_part (view source)

Defined in: uf_part.h

Overview
Given a part determine whether it comes from a JT file, whether that JT file exists and whether that JT file contains brep data.

Return
Return code:
0 = Success
> 0 = Error code, use UF_get_fail_message to obtain error message string

Environment
Internal and External

History
Released in NX4

Required License(s)
gateway

```
int UF_PART_ask_jt_info_of_part
(
    const tag_t part_tag,
    logical * from_jt_file,
    logical * jt_file_exists,
    logical * contains_breps
)
```

const tag_t	**part_tag**	Input	The tag of the part to query
logical *	**from_jt_file**	Output	True if part file comes from a JT file
logical *	**jt_file_exists**	Output	True if the JT file exists
logical *	**contains_breps**	Output	True if the file contains brep data

---

UF_PART_ask_last_modified_version (view source)

Defined in: uf_part.h

Overview
Returns the version number of last significant change (aka major version) for the given part.

NOTE: The version number returned is not the index into the part

history list for the last significant change. To get the index into the part history list for the last significant change, ask the number of part histories in the list, and then subtract the modified version number.

Environment
Internal and External

See Also
UF_PART_ask_minor_version
UF_PART_ask_part_history
UF_PART_ask_nth_history
UF_PART_ask_num_histories
UF_PART_create_history_list
UF_PART_delete_history_list

History
Originally released in NX302 and NX4

Required License(s)
gateway

```
int UF_PART_ask_last_modified_version
(
    tag_t part,
    int* modified_version
)
```

tag_t	**part**	Input	Tag of part in question
int*	**modified_version**	Output	Version number were part was last modified

---

UF_PART_ask_minor_version (view source)

Defined in: uf_part.h

Overview
Returns the minor version number for the given part.
Use UF_PART_ask_last_modified_version instead if you need the major version.

NOTE: The version number returned is not the index into the part history list for the last significant change. To get the index into the part history list for the last significant change, ask the number of part histories in the list, and then subtract the minor version number.

Environment
Internal and External

See Also
UF_PART_ask_part_history
UF_PART_ask_nth_history
UF_PART_ask_num_histories
UF_PART_create_history_list
UF_PART_delete_history_list

History
Originally released in NX404, NX501, and NX6

Required License(s)
gateway

```
int UF_PART_ask_minor_version
(
    tag_t part,
    int* minor_version
)
```

tag_t	**part**	Input	Tag of part in question
int*	**minor_version**	Output	Minor version number were part was last modified or saved

---

UF_PART_ask_nth_history (view source)

Defined in: uf_part.h

Overview
Returns the information about a particular save history from the given history list object.

The history_list parameter must be a UF_PART_history_list_p_t variable which has been created by UF_PART_create_history_list and filled in with UF_PART_ask_part_history.

The index parameter defines which "save" is being inquired upon and may range from 0 to num - 1, where num is the number of histories as returned by UF_PART_ask_num_histories. The histories are stored in reverse chronological order (most recent history first, at index=0).

The first history is also special in that it represents when the part was loaded in this session, as opposed to the other histories which represent when the part was saved.

The program, user, and machine variables have stored in them the addresses of strings (char variables) representing the program, user, and machine type for the requested "save" of the part. These addresses are actually the addresses within the history_list object of these values and should NOT be modified or freed by the caller of this routine. If you wish to modify the values returned for these variables, use strcpy to copy them and then manipulate the copies.

Environment
Internal and External

See Also
UF_PART_ask_part_history
UF_PART_ask_num_histories
UF_PART_clear_history_list
UF_PART_create_history_list
UF_PART_delete_history_list
Refer to the example

Required License(s)
gateway

```
int UF_PART_ask_nth_history
(
    UF_PART_history_list_p_t history_list,
    int index,
    char * * program,
    char * * user,
    char * * machine,
    int * version,
    int * gmtime
)
```

UF_PART_history_list_p_t	**history_list**	Input	Address of a history list object which contains the save information for a part (obtained by UF_PART_ask_part_history).
int	**index**	Input	Index (starting at zero) of the particular part save for which information is requested.
char * *	**program**	Output	Address of a pointer to a character (char) variable into which is stored the address of the name of the program which saved this version of the part.
char * *	**user**	Output	Address of a pointer to a character (char) variable into which is stored the address of the name of the user who saved this version of the part.
char * *	**machine**	Output	Address of a pointer to a character (char) variable into which is stored the address of the name of the machine type on which this version of the part was saved.
int *	**version**	Output	Address of an int variable into which is stored the version number associated with this version of the part.
int *	**gmtime**	Output	Address of an int variable into which is stored the time associated with the requested "save" of the part. This value is the number of seconds since January 1, 1970 12:00 AM GMT. See the routines in the standard UNIX time.h include file for routines to manipulate and display time values.

UF_PART_ask_nth_part (view source)

Defined in: uf_part.h

Overview

Returns the tag of the "n'th" part in the session. The parts are considered to be numbered 0 to (UF_PART_ask_num_parts()-1). If the "n'th" part doesn't exist, a NULL_TAG is returned.

Return

returns a part tag or a NULL_TAG if the part does not exist

Environment

Internal and External

Required License(s)

gateway

tag_t UF_PART_ask_nth_part
(
int part_num
)

int	**part_num**	Input	part number on which to inquire tag.

UF_PART_ask_num_histories (view source)

Defined in: uf_part.h

Overview
Returns the number of save histories in the given history list object.

Environment
Internal and External

See Also
UF_PART_ask_part_history
UF_PART_ask_nth_history
UF_PART_clear_history_list
UF_PART_create_history_list
UF_PART_delete_history_list
Refer to the example

Required License(s)
gateway

int UF_PART_ask_num_histories
(
UF_PART_history_list_p_t history_list,
int * number
)

UF_PART_history_list_p_t	**history_list**	Input	Address of a history list object which contains the save information for a part (obtained by UF_PART_ask_part_history).
int *	**number**	Output	Address of an int variable into which is stored the number of save histories in the given history list.

UF_PART_ask_num_parts (view source)

Defined in: uf_part.h

Overview
Returns the number of parts loaded in the current session.

Return
returns an integer value representing the number of parts loaded in the current session.

Environment
Internal and External

Required License(s)
gateway

```
int UF_PART_ask_num_parts
(
    void
)
```

UF_PART_ask_part_history (view source)

Defined in: uf_part.h

Overview
Fills in a history list object with the save history of the given part.

Environment
Internal and External

See Also
UF_PART_ask_nth_history
UF_PART_ask_num_histories
UF_PART_clear_history_list
UF_PART_create_history_list
UF_PART_delete_history_list
Refer to the example

Required License(s)
gateway

```
int UF_PART_ask_part_history
(
    tag_t part,
    UF_PART_history_list_p_t history_list
)
```

tag_t	**part**	Input	Tag of the part for which save history information is requested.
UF_PART_history_list_p_t	**history_list**	Input	Address of a history list object into which is stored the save information for the given part.

UF_PART_ask_part_history_with_rename_info (view source)

Defined in: uf_part.h

Overview
Fills in a history list object with the save history of the given part.
This will also include history list items for each time the part was renamed.

Environment
Internal and External

See Also
UF_PART_ask_nth_history
UF_PART_ask_num_histories
UF_PART_clear_history_list
UF_PART_create_history_list
UF_PART_delete_history_list
Refer to the example

Required License(s)
gateway

```
int UF_PART_ask_part_history_with_rename_info
(
    tag_t part,
    UF_PART_history_list_p_t history_list
)
```

tag_t	**part**	Input	Tag of the part for which save history information is requested.
UF_PART_history_list_p_t	**history_list**	Input	Address of a history list object into which is stored the save information for the given part.

UF_PART_ask_part_name (view source)

Defined in: uf_part.h

Overview
Returns the file specification associated with a part tag.
NOTE: Do not use a part occurrence tag from an assembly as input to this function. See UF_ASSEM_ask_part_name_of_child, in order to obtain the part name of an occurrence.

Return
Return code:
0 = Success
not 0 = Error code, use UF_get_fail_message to obtain error message string

Environment
Internal and External

See Also
See UF_ASSEM_ask_part_name_of_child
in order to obtain the part name of an occurrence.

Required License(s)
gateway

int UF_PART_ask_part_name
(
tag_t part,
char part_fspec [MAX_FSPEC_BUFSIZE]
)

tag_t	**part**	Input	Tag of part
char	**part_fspec [MAX_FSPEC_BUFSIZE]**	Output	file specification of part

---

UF_PART_ask_part_tag (view source)

Defined in: uf_part.h

Overview
Returns the part tag associated with the part of the specified name.
The full path name is not required. Use CLI Format name for TCIN mode.
Foreign names will be mapped to their corresponding NX part name.

Note: In case of Managed mode if Multiple Revisions is OFF then
this method will return the part tag of any loaded revision of the passed in item id.
If Multiple Revisions is ON then this method will return the part tag of the exact revision passed into this method.
If the given part revision is not loaded in NX session then this will return a NULL tag when Multiple Revisions is enabled.

Return
returns a part tag or a NULL_TAG if the part is not
loaded in the current session.

Environment
Internal and External

Required License(s)
gateway

tag_t UF_PART_ask_part_tag
(
const char * part_name
)

const char *	**part_name**	Input	name of part to inquire the tag of

---

UF_PART_ask_status (view source)

Defined in: uf_part.h

Overview
Returns the integer that represents the status of the part. This status value is contained in the part file's header. Unless the status has been set, it's default value is zero (0).

Environment
Internal and External

See Also
UF_PART_set_status

Required License(s)
gateway

```
int UF_PART_ask_status
(
    tag_t part_tag,
    int * status
)
```

tag_t	**part_tag**	Input	Tag of the part
int *	**status**	Output	Status of the part.

UF_PART_ask_template_filename (view source)

Defined in: uf_part.h

Overview
Returns the leaf name of the template part of the input family instance. Returns an empty string if the part is not a family instance.

Environment
Internal and External

History
Prior to NX3 this routine returned the display name instead of the leaf name. In NX3 and beyond, use UF_PART_file_name_for_display() on the leaf name to get to the display name.

Required License(s)
gateway

```
int UF_PART_ask_template_filename
(
    tag_t part,
    char template_name [ MAX_FSPEC_BUFSIZE ]
)
```

tag_t	**part**	Input	Tag of a family instance part

char	**template_name [MAX_FSPEC_BUFSIZE]**	Output	file specification of the associated template

---

UF_PART_ask_units (view source)

Defined in: uf_part.h

Overview
Returns a value indicating whether the part is in English or metric units.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
gateway

```
int UF_PART_ask_units
(
    tag_t part,
    int * part_units
)
```

tag_t	**part**	Input	Tag of the part to ask units of
int *	**part_units**	Output	Units of Part: UF_PART_METRIC UF_PART_ENGLISH

UF_PART_check_part_writable (view source)

Defined in: uf_part.h

Overview
Determine if the current user has write access to a part.

Environment
Internal and External

History
Released in NX3.0.3 and NX4.0.0

Return
0 = Success
Otherwise = Error code, use UF_get_fail_message to obtain error message string

Required License(s)
gateway

```
int UF_PART_check_part_writable
(
    const char* part_name,
    logical * writable
)
```

const char*	**part_name**	Input	Name of the part to query the write status of
logical *	**writable**	Output	TRUE - You have write access to the part. FALSE - You do not have write access to the part.

UF_PART_cleanup (view source)

Defined in: uf_part.h

Overview

Enables you to specify one or all of the part cleanup procedures. Each part cleanup procedure has a definitions which represents a bit mask. You OR these bits together to obtain the option_mask argument. The part cleanup procedure for each bit is as follows.

UF_PART_cleanup_highlight - In internal Open API, turns off object highlighting for all the displayable objects in the root part.
NOTE: this will not remove objects from the selection list, it simply unhighlights everything.
In external Open API programs, this option does not do anything.

UF_PART_cleanup_all_groups - Deletes all group objects of the work part that have no members.

UF_PART_cleanup_unnamed_groups - Deletes all unnamed groups of the work part that have no members.

UF_PART_cleanup_unreferenced - Deletes all unreferenced, condemned objects in the work part that have a reference count of zero and that have no back links. Condemned objects with only one back link are deleted provided the single back link is the view dependent link from the object to it's dependent view.

UF_PART_cleanup_feature - Performs Solids Clean-Up. The system analyzes all solid bodies to ensure that all of their faces and edges are present and consistent with the Parasolid representation. If an error is detected the system attempts to repair the part.

UF_PART_cleanup_all - Sets all bits, so that all part cleanup functions are run on all parts. To only run on the work part you must clear the bits UF_PART_cleanup_parts_all and UF_PART_cleanup_parts_components by specifying UF_PART_cleanup_all & ~UF_PART_cleanup_parts_all &

~UF_PART_cleanup_parts_components

UF_PART_cleanup_parts_all - Cleans up all fully loaded parts in the session.

UF_PART_cleanup_parts_components - Cleans up the work part, and if the work part is an assembly, it cleans up the components of the work assembly.

UF_PART_cleanup_spreadsheet - Deletes all spreadsheets from the part file.

UF_PART_cleanup_mating - Performs Mating Condition Clean-Up. The system analyzes all mating conditions to ensure that all their data is consistent. If an error is detected the system attempts to repair it.
Note: this operation will load referenced parts.

UF_PART_cleanup_CAM_cleanup - Deletes orphan tool paths and deletes any of the following that are unreferenced: parameter sets, CAM geometry entities, post processor command sets, and non-cutting move data.

UF_PART_cleanup_fonts - Eliminates unused character fonts from a part.

UF_PART_cleanup_unreferenced_exps - Deletes all unreferenced (unused) expressions in the part. Also deletes all expressions that are only referenced by unreferenced expressions.
For example:
a=10
b=a
Both a and b are deleted.

UF_PART_cleanup_occurrences - Removes redundant changes in all occurrences having same display properties as their prototypes.

UF_PART_cleanup_visual_editor - Deletes all visual editor data from the part file.

UF_PART_cleanup_hwo_force_demoting - Removes changes from all occurrences even if their display display properties are different from their prototypes.
Beware UF_PART_cleanup_hwo_force_demoting is mutually exclusive with UF_PART_cleanup_occurrences.

UF_PART_cleanup_drafting - Part cleanup on drafting objects / drawings.

UF_PART_cleanup_sketch - Fix off-plane sketch curves

UF_PART_cleanup_delete_broken_interpart - Delete all inter-part links where one or more components in that inter-part link have been removed.

UF_PART_cleanup_delete_all_materials - Deletes all materials from the part file.

UF_PART_cleanup_delete_duplicate_lights - Deletes all duplicate light source objects from the part file.
A light source object is considered a duplicate if it has the same name as another light source object in the part.

UF_PART_cleanup_delete_invalid_attributes - Deletes all invalid user attributes from the part file.

Environment
Internal and External

Required License(s)
gateway

int UF_PART_cleanup
(
unsigned int option_mask
)

unsigned int	**option_mask**	Input	Bit mask that specifies the particular cleanup procedures to perform.

UF_PART_clear_history_list (view source)

Defined in: uf_part.h

Overview
Clears the information contained within the specified history list object (version number, time, user, program, etc.) such that this object can be reused with a UF_PART_ask_part_history call with a different part.

Environment
Internal and External

See Also
UF_PART_ask_part_history
UF_PART_ask_nth_history
UF_PART_ask_num_histories
UF_PART_create_history_list
UF_PART_delete_history_list
Refer to the example

Required License(s)
gateway

int UF_PART_clear_history_list
(
UF_PART_history_list_p_t historiy_list
)

UF_PART_history_list_p_t	**historiy_list**	Input	Address of a history list structure allocated by UF_PART_create_history_list

---

UF_PART_close (view source)

Defined in: uf_part.h

Overview

Closes the selected part, and optionally, all parts under it in the assembly tree. If you run this routine in internal mode, the system puts up a confirmation window asking you if the modified part should be closed. If you run this routine in external mode, the system assumes the part should be closed.

Closing a part with UF_PART_close does not remove undo marks in NX. Therefore, it is possible to consume large amounts of memory if you use this function. To free this memory, call UF_UNDO_delete_all_marks after you call UF_PART_close.

Return

Return code:
0 = Success
<0 = CFI Status Code
1 = Modified part not saved (warning status)
2 = Part not root of assem tree
> 2 = Error code, use UF_get_fail_message to obtain error message string.

Environment

Internal and External

Required License(s)

gateway

```
int UF_PART_close
(
    tag_t part,
    int scope,
    int mode
)
```

tag_t	**part**	Input	part object identifier
int	**scope**	Input	scope specifies how much of the part to close. 0 = Only specified part 1 = Part and all sub-assemblies
int	**mode**	Input	mode 0 = Ask confirmation if part is modified (Internal only, External assumes "Yes, Delete" answer) 1 = Unload part(s) even if modified 2 = Unload part(s) only if not modified

---

UF_PART_close_all (view source)

Defined in: uf_part.h

Overview

Closes all parts in the current session.

The closure of all parts in NX results in a return to the NX No Part environment. When NX transitions to a no part environment, all dialogs are automatically closed in order to secure a safe, clean environment with no retained part dependencies such as selected objects or assumptions that parts are still open.

Note that any block of code that attempts to display a user interface such as a dialog, file selection box or message box will not be created if it follows a call to UF_UI_close_parts and resides in the same path of execution. Also note that ANY code following a request to display any type of user interface will NOT be executed due to the change back to the no part application.

For example,

```
UF_PART_close_all();
UF_print_syslog("This line of code WILL be executed\n", FALSE);

UF show any dialog(); // This dialog will not be displayed
UF_print_syslog("This line of code will not be executed\n", FALSE);
```

A separate user function must be invoked in order to create a dialog after all parts are closed and NX enters the no part application.

Environment
Internal and External

History
V15.0 change: This function was modified to return an integer error code.

Required License(s)
gateway

```
int UF_PART_close_all
(
    void
)
```

UF_PART_close_cset (view source)

Defined in: uf_part.h

Overview
Closes all the components in the given component set, and removes them from the Current Components cset. Any part which is no longer required by other components in any loaded assembly is then closed.

Environment
Internal and External

See Also
UF_PART_open_cset
Refer to the example

Required License(s)
gateway

```
int UF_PART_close_cset
(
    tag_t cset
)
```

tag_t	**cset**	Input	set of components to close

UF_PART_create_family_instance (view source)

Defined in: uf_part.h

Overview

Creates a new part containing an instance of a part family member. The new part is NOT saved and/or closed during this operation. These can be done separately on the new part using UF_ASSEM_set_work_part_context_quietly, UF_PART_save, UF_ASSEM_restore_work_part_context_quietly and UF_PART_close.

Environment

Internal and External

See Also

UF_PART_update_family_instance

Required License(s)

solid_modeling

```
int UF_PART_create_family_instance
(
    tag_t family,
    int member_index,
    tag_t * part,
    tag_t * instance
)
```

tag_t	**family**	Input	Object identifier of the family.
int	**member_index**	Input	Index of the family member.
tag_t *	**part**	Output	Object identifier of the part containing the family instance
tag_t *	**instance**	Output	Tag of the family instance.

UF_PART_create_history_list (view source)

Defined in: uf_part.h

Overview

Creates a structure for use by other Part History routines and returns the address of this history list object.

Environment
Internal and External

See Also
UF_PART_ask_part_history
UF_PART_ask_nth_history
UF_PART_ask_num_histories
UF_PART_clear_history_list
UF_PART_delete_history_list
Refer to the example

Required License(s)
gateway

int UF_PART_create_history_list
(
UF_PART_history_list_p_t * history_list
)

UF_PART_history_list_p_t *	**history_list**	Output to UF_*free*	Address of a structure allocated by this routine which may be used by other history routines to obtain a part's save histories. Should be freed using UF_PART_delete_history_list.

UF_PART_delete_history_list (view source)

Defined in: uf_part.h

Overview
Deletes (frees) the given history list object (allocated by the UF_PART_create_history_list routine). If save audit information for a part has been added to this structure by UF_PART_ask_part_history, this data is also freed.

Environment
Internal and External

See Also
UF_PART_ask_part_history
UF_PART_ask_nth_history
UF_PART_ask_num_histories
UF_PART_clear_history_list
UF_PART_create_history_list
Refer to the example

Required License(s)
gateway

int UF_PART_delete_history_list
(
UF_PART_history_list_p_t history_list
)

UF_PART_history_list_p_t	**history_list**	Input	Address of a history list structure allocated by UF_PART_create_history_list.

UF_PART_evaluate_write_state (view source)

Defined in: uf_part.h

Overview
This function will check to see if a part is fully loaded. If it is then the write state is checked and the Assembly Navigator will be updated. This will allow the Assembly Navigator icons to update without closing and reopening the part file.

Environment
Internal and External

History
Released in NX3

Required License(s)
gateway

int UF_PART_evaluate_write_state
(
tag_t part_tag
)

tag_t	**part_tag**	Input	tag of part file to check status

UF_PART_export (view source)

Defined in: uf_part.h

Overview
Exports the specified objects to the specified part. The objects are copied into the destination part. Calling this function is equivalent to calling UF_PART_export_with_options with the following options:
new_part = true
params_mode = UF_PART_maintain_params
expression_mode = UF_PART_copy_exp_deeply

See UF_PART_export_with_options for more details on the behavior of this function and for a description of the return value.

Environment
Internal and External

See Also
UF_PART_export_with_options
Refer to the example

History

For V11.0, the behavior of this function was modified to retain feature parameters in the transferred objects. Prior to V11.0, feature parameters were removed.

For V14.0, the behavior of this function was modified as follows: If you are exporting geometry to a part which is loaded (or partially loaded) in a session, then that part is no longer saved and closed as a consequence of calling this routine. The destination part is only saved and closed as part of this operation if it was not loaded beforehand. This makes the Open API behavior consistent with the interactive behavior.

For V14.0.1, the behavior of this function was modified as follows: When exporting geometry using this function, the geometry would not be oriented properly in the destination part if the WCS of the source part was away from the origin. This is now fixed so that the geometry is transformed from the WCS of the source part to the WCS of the destination part. The Open API behavior is now the same as the interactive behavior.

Required License(s)

gateway

```
int UF_PART_export
(
    const char * part_name,
    int num_objects,
    tag_t object_array [ ]
)
```

const char *	**part_name**	Input	Name of part to export objects to.
int	**num_objects**	Input	Number of objects in object_array to be exported.
tag_t	**object_array [ ]**	Input	Objects to be exported.

UF_PART_export_with_options (view source)

Defined in: uf_part.h

Overview

Exports the specified objects to the specified part. The objects are copied into the destination part. Any other transferable objects upon which the given objects depend are also exported.
Note that the export operation fails if there is no display part.

NOTE: Object occurrences and part occurrences are not exportable.

If an object depends on another object which is not exportable, then that object is not exported and a warning value of UF_PART_warn_objects_not_copied is returned. However, the operation continues, and other requested exportable objects are still exported. This situation can arise in two circumstances.

1. A drafting object is not exported if its associated geometry is not exported.

2. Datums cannot be exported if feature parameters are removed.

If any other error occurs, the operation does not succeed and the appropriate error code is returned.
The options argument controls the behavior of this operation:
Refer to the table

Environment

Internal and External

See Also

UF_PART_export
Refer to the example

History

For V14.0, the behavior of this function was modified as follows:
If you are exporting geometry to a part which is loaded (or partially loaded) in a session, then that part is no longer saved and closed as a consequence of calling this routine. The destination part is only saved and closed as part of this operation if it was not loaded beforehand. This makes the Open API behavior consistent with the interactive behavior.

For V14.0.1, the behavior of this function was modified as follows:
When exporting geometry using this function, the geometry would not be oriented properly in the destination part if the WCS of the source part was away from the origin. This is now fixed so that the geometry is transformed from the WCS of the source part to the WCS of the destination part. The Open API behavior is now the same as the interactive behavior.

For V15.0, this function was modified as follows:
The UF_PART_maintain_all_params enumerated constant was added to UF_PART_export_params_mode_t.

Required License(s)

gateway

```
int UF_PART_export_with_options
(
    const char * part_name,
    int num_objects,
    tag_t object_array [ ] ,
    UF_PART_export_options_p_t options
)
```

const char *	**part_name**	Input	Name of part to export objects to.
int	**num_objects**	Input	Number of objects in object_array to be exported.
tag_t	**object_array [ ]**	Input	Objects to be exported.
UF_PART_export_options_p_t	**options**	Input	Pointer to struct containing options to control how the objects are exported.

UF_PART_file_name_for_display (view source)

Defined in: uf_part.h

Overview
Converts the given internal part name into a display name in the same format as would be used by NX for menus, banners and etc. This function accepts a CLI format name as well as an internal format name, for NX Manager names.
e.g., From: "@DB/peters-part/A/spec/sheet1" (cli format)
or
From: "%UGMGR=3.2 PN=Peters..." (internal format)
To: "peters-part/A (spec: sheet1)" (display format)

Environment
Internal and External

See Also
See the example provided with UF_UGMGR_encode_part_filename

History
This function was originally released in NX 5.0 and is mandatory if Longer IDs functionality is enabled NX/Manager

Required License(s)
gateway

```
int UF_PART_file_name_for_display
(
    const char * name_format,
    char * * display_name
)
```

const char *	**name_format**	Input	Command Line Input format, e.g., "@DB/peters-part/A/spec/sheet1" or Internal format "%UGMGR=3.2 PN=Peters..."
char * *	**display_name**	Output to UF_*free*	Display format "peters-part/A (spec: sheet1)"

UF_PART_find_family_instance (view source)

Defined in: uf_part.h

Overview
Returns the name of the part which contains the part family instance if the part can be found.

Environment
Internal and External

Required License(s)
gateway

```
int UF_PART_find_family_instance
(
    tag_t family,
    int member_index,
    logical load,
    logical use_load_options,
    char * * part_name
)
```

tag_t	**family**	Input	Object identifier of the family.
int	**member_index**	Input	Index of the family member.
logical	**load**	Input	Load part if found?
logical	**use_load_options**	Input	Use load options to find part?
char * *	**part_name**	Output to UF_*free*	Name of part containing the family instance. This should be freed by calling UF_free.

UF_PART_free_load_status (view source)

Defined in: uf_part.h

Overview
Routine to free the load status structure.

Environment
Internal and External

History
Released in NX3

Required License(s)
gateway

```
int UF_PART_free_load_status
(
    UF_PART_load_status_p_t load_status
)
```

UF_PART_load_status_p_t	**load_status**	Input / Output	load status to be freed

---

UF_PART_import (view source)

Defined in: uf_part.h

Overview

Merges an NX part or Solid Edge part (.par or .psm file extension) from disk into the current Work Part. Inputs are specified in the "modes" structure. Solid Edge parts are imported as unparameterized solids.

See the description of the modes structure .

Return

Return code:
0 = Success
< 0 = CFI Status Code
> 0 = Error code, use UF_get_fail_message to obtain error message string

Environment

Internal and External

History

In V15.0, this function was enhanced so that it could import Solid Edge parts.

Required License(s)

gateway

```
int UF_PART_import
(
    const char * file_name,
    UF_import_part_modes_t * modes,
    double dest_csys [ 6 ] ,
    double dest_point [ 3 ] ,
    double scale,
    tag_t * group
)
```

const char *	**file_name**	Input	File specification of NX disk file to import.
UF_import_part_modes_t *	**modes**	Input	Inputs are specified in the `modes' structure.
double	**dest_csys [6]**	Input	Destination Coordinate System. dest_csys[0..2] is an X-direction vector, and dest_csys[3..5] is a y direction vector. These are used as input to UF_MTX3_initialize to create the full coordinate system matrix.
double	**dest_point [3]**	Input	Destination point of imported part
double	**scale**	Input	The scale size for the imported part.
tag_t *	**group**	Output	If grouping is desired, the group is returned, otherwise it is a NULL_TAG

UF_PART_import_xt_hidden (view source)

Defined in: uf_part.h

Overview

Given x_t file name, import all its bodies in NX and hide them

Assumptions:

1. It imports the x_t into the current work part. It will return error if there is no workpart.

Note:
User need to free the bodyTags array

Environment

Internal and External

Return

0 = Success
Otherwise = Error code, use UF_get_fail_message to obtain error message string

Required License(s)

gateway

```
int UF_PART_import_xt_hidden
(
    const char* xtFileName,
    int* numBodies,
    tag_t* * bodyTags
)
```

const char*	**xtFileName**	Input	Input xt Filename
int*	**numBodies**	Output	Count of bodies
tag_t* *	**bodyTags**	Output to UF_*free*	An allocated array containing the tags of created bodies. This must be freed by calling UF_free.

UF_PART_is_family_inst_current (view source)

Defined in: uf_part.h

Overview

Queries whether a part family instance is current. Returns TRUE if a part family instance is up to date with respect to its template part.

Environment

Internal and External

Required License(s)

gateway

```
int UF_PART_is_family_inst_current
(
    tag_t part,
    logical * is_inst_current
```

)

tag_t	**part**	Input	Tag of the part. If you have a part occurrence, use UF_ASSEM_ask_prototype_of_occ to get the tag of the part to pass into this routine. If you have a part instance, use UF_ASSEM_ask_parent_of_instance to get the tag of the part to pass into this routine.
logical *	**is_inst_current**	Output	Flag indicating result.

UF_PART_is_family_instance (view source)

Defined in: uf_part.h

Overview
Queries if a part contains a part family instance.

Environment
Internal and External

Required License(s)
gateway

int UF_PART_is_family_instance
(
tag_t part,
logical * is_family_instance
)

tag_t	**part**	Input	Tag of the part. If you have a part occurrence, use UF_ASSEM_ask_prototype_of_occ to get the tag of the part to pass into this routine. If you have a part instance, use UF_ASSEM_ask_parent_of_instance to get the tag of the part to pass into this routine.
logical *	**is_family_instance**	Output	Flag indicating result.

UF_PART_is_family_template (view source)

Defined in: uf_part.h

Overview
Queries if a part contains a part family.

Environment
Internal and External

Required License(s)

gateway

int UF_PART_is_family_template
(
tag_t part,
logical * is_family_template
)

tag_t	**part**	Input	Tag of the part. If you have a part occurrence, use UF_ASSEM_ask_prototype_of_occ to get the tag of the part to pass into this routine. If you have a part instance, use UF_ASSEM_ask_parent_of_instance to get the tag of the part to pass into this routine.
logical *	**is_family_template**	Output	Flag indicating result. TRUE = Part is a family template

---

UF_PART_is_loaded (view source)

Defined in: uf_part.h

Overview

Performs a query in the current session on the load status of a part. Parts which are partially loaded can be fully loaded with UF_PART_open.

Return

return code:
0 =Part is not loaded.
1 =Part is fully loaded in session.
2 =Part is partially loaded in session.
Other =Error Code.

Environment

Internal and External

Required License(s)

gateway

int UF_PART_is_loaded
(
const char * part_name
)

const char *	**part_name**	Input	name of part

UF_PART_is_modified (view source)

Defined in: uf_part.h

Overview
Inquires whether a part is loaded and modified in the current session.
Returns a logical value based on the results of the inquiry.

Return
Returns TRUE if the part is loaded and modified in
the current session,
FALSE otherwise.

Environment
Internal and External

Required License(s)
gateway

```
logical UF_PART_is_modified
(
    tag_t part
)
```

tag_t	**part**	Input	tag of part to inquire about

UF_PART_new (view source)

Defined in: uf_part.h

Overview
Create a new NX part in the current session and makes it the work part.

Environment
Internal and External

Required License(s)
gateway

```
int UF_PART_new
(
    const char * part_name,
    int units,
    tag_t * part
)
```

const char *	**part_name**	Input	Name of the new part. This part name must be unique for the session. The leaf name for two parts in the same session can not be the same.
int	**units**	Input	Specifies either English or Metric units. 1 = Metric 2 = English
tag_t *	**part**	Output	Tag of the created part or NULL_TAG if there is an error.

UF_PART_open (view source)

Defined in: uf_part.h

Overview

Retrieves an existing NX part or Solid Edge part into the session and makes it the work and display part. Solid Edge parts (.par, .psm, .pwd or .asm file extension) are opened by extracting the Parasolids data from the Solid Edge part and then importing this data into a new NX part with a .prt extension. The file name of the new NX part has the Solid Edge part name and a ".prt" file extension. If there is an existing NX part with the same name as the Solid Edge part, then this function returns an error.

The NX part file that you create by opening a Solid Edge part file contains one or more unparameterized solid bodies.

Other files can be opened with this call. The following extensions are valid - .udf, .bkm, .xpk and .jt. Foreign files with the following extensions can also be opened with UF_PART_open - .igs, .stp, .dxf, .dwg and .model.

If a part is a read-only part, and the part retrieval modifies the part, you will receive the error code UF_PART_err_read_only_modified, for that part when you call UF_PART_open. In this case UF_PART_open will not return a zero status.

Environment

Internal and External

History

In V15.0, this function was enhanced so that it could open Solid Edge parts.

Required License(s)

gateway

```
int UF_PART_open
(
    const char * part_name,
    tag_t * part,
    UF_PART_load_status_t * error_status
)
```

const char *	**part_name**	Input	Name of part to retrieve.
tag_t *	**part**	Output	The tag of the retrieved part or NULL_TAG if part retrieval fails.
UF_PART_load_status_t *	**error_status**	Output to UF_*free*	The user allocated structure <error_status> is filled with the names and associated error codes of any parts that did not load correctly. The structure must be freed with UF_PART_free_load_status.

For details see the definition
of UF_PART_load_status_t.

UF_PART_open_component_as (view source)

Defined in: uf_part.h

Overview

Opens a component with a different name other than the one which it was saved with. As long as the component part is the same, the component is opened without loss of assembly level associativity. If you want to open the component with an unrelated component part you need to "allow substitutions", using the UF_ASSEM_set_assem_options function.

This function is useful in cases where component parts have been renamed without their parent assemblies being open to register the new names.

The component must be unloaded and its parent assembly must be loaded to start with. All occurrences of the component in the assembly are opened.

It is not possible to open a component with a name of a component that is already used in the assembly. For this reason you need to ensure that all assembly levels that need to refer to the newly named components are open at the time when the component is opened, so that they register the new name of the component simultaneously. Because this can be an easy operation to get wrong, it is recommended that you maintain a copy of the original assembly until you have verified that the assembly is correctly renamed, and can be opened with the newly named components.

Return

Return code:
0 = No error
1 = Parent not loaded, or component has no parent
2 = New part is not substitutable for old part
3 = Old part is already loaded
4 = New part is already loaded
5 = New part is already referenced elsewhere in there assembly
n = Error code

Environment

Internal and External

See Also

UF_ASSEM_ask_assem_options
UF_ASSEM_set_assem_options

Required License(s)

gateway

int UF_PART_open_component_as
(

```
    tag_t component,
    const char * old_name,
    const char * new_name,
    tag_t * part,
    UF_PART_load_status_t * error_status
)
```

tag_t	**component**	Input	The component being opened
const char *	**old_name**	Input	The component's current part name with ".prt" file extension.
const char *	**new_name**	Input	The component's new name part name with ".prt" file extension.
tag_t *	**part**	Output	The component part that is opened
UF_PART_load_status_t *	**error_status**	Output to UF_*free*	The user allocated structure <error_status> is filled with the names and associated error codes of any parts that did not load correctly. The structure must be freed with UF_PART_free_load_status. For details see the definition of UF_PART_load_status_t.

UF_PART_open_cset (view source)

Defined in: uf_part.h

Overview

Opens all the unloaded parts for the components in the given cset. All the components are added to the CurrentComponents cset. All parts between the root part and required parts are opened, so in the case of the Car example from the Assemblies chapter, if the left front wheel is in the component set, the axle is opened, and the front axle component is added to the CurrentComponents cset.

Return

Return code:
0 = No error
not 0 = Error code of first part in load status which failed to load.

Environment

Internal and External

See Also

UF_PART_close_cset
Refer to the example

Required License(s)

gateway

```
int UF_PART_open_cset
(
    tag_t cset,
    UF_PART_load_status_t * load_status
)
```

tag_t	**cset**	Input	set of components to open
UF_PART_load_status_t *	**load_status**	Output to UF_*free*	The user allocated structure <error_status> is filled with the names and associated error codes of any parts that did not load correctly. The structure must be freed with UF_PART_free_load_status. For details see the definition of UF_PART_load_status_t.

UF_PART_open_quiet (view source)

Defined in: uf_part.h

Overview

Retrieves an existing NX part or Solid Edge part into the session without making it the work and display part. Solid Edge parts (.par or .psm file extension) are opened by extracting the Parasolids data from the Solid Edge part and then importing this data into a new NX part with a .prt extension. The file name of the new NX part has the Solid Edge part name and a ".prt" file extension. If there is an existing NX part with the same name as the Solid Edge part, then this function returns an error.

Environment

Internal and External

History

In V15.0, this function was enhanced so that it could open Solid Edge parts.

Required License(s)

gateway

```
int UF_PART_open_quiet
(
    const char * part_name,
    tag_t * part,
    UF_PART_load_status_t * error_status
)
```

const char *	**part_name**	Input	Name of part to retrieve.
tag_t *	**part**	Output	The tag of the retrieved part or NULL_TAG if part retrieval fails.
UF_PART_load_status_t *	**error_status**	Output to UF_*free*	The user allocated structure <error_status> is filled with the names and associated error codes of any parts

	that did not load correctly. The structure must be freed with UF_PART_free_load_status. For details see the definition of UF_PART_load_status_t.

UF_PART_open_single_component_as (view source)

Defined in: uf_part.h

Overview

Opens a single occurrence of a component, using a different part name than the part name it has been stored with. The component being opened-as must be unloaded. The new part being substituted for it can either be loaded or not. Both components must have identical uid's. The user allocated structure <error_status> will be filled with the name and associated error code of the loaded part. The allocated arrays must be freed.

Return

Return code:
0 = success
1 = Parent not fully loaded, or component has no parent
2 = New part can't be loaded
3 = Old part is already loaded
4 = New part not substitutable for old part (uid's must match)
n = Unspecified error

If the part passed in for replacement does not share a uid with the original component then an "Illegal Instruction" error is returned.

Environment

Internal and External

See Also

UF_ASSEM_ask_assem_options
UF_ASSEM_set_assem_options

Required License(s)

gateway

```
int UF_PART_open_single_component_as
(
    tag_t component,
    const char * new_part_name,
    tag_t * part,
    UF_PART_load_status_t * error_status
)
```

tag_t	**component**	Input	The component being opened
const char *	**new_part_name**	Input	The component's new part name
tag_t *	**part**	Output	The component part that is opened

UF_PART_load_status_t *	**error_status**	Output to UF_*free*	The user allocated structure <error_status> is filled with the names and associated error codes of any parts that did not load correctly. The allocated arrays must be freed with UF_free_string_array and UF_free(). For details see the definition of UF_PART_load_status_t.

UF_PART_rename (view source)

Defined in: uf_part.h

Overview

Changes the name of the part specified by the tag to the name given. It changes that part only and does not flag as modified any parent parts in the session for which this part may be a component.

Environment

Internal and External

Required License(s)

gateway

```
int UF_PART_rename
(
    tag_t part_tag,
    const char * new_part_name
)
```

tag_t	**part_tag**	Input	tag of part to be renamed
const char *	**new_part_name**	Input	new name for part

UF_PART_reopen (view source)

Defined in: uf_part.h

Overview

Unloads and reloads a part that is different on disk (has been modified on disk or in the session). If the scope flag is set to do subassemblies, then all parts in the given part that are different on disk are reopened. Parts that are up to date with what is on disk are skipped. If the mode flag is set to not reopen parts that are modified in the session, then parts that have been modified in the session are skipped. A mode value of 0 is only valid when using internal Open API. If this option is used, then a confirmation dialog comes up for parts that have been modified in the session to confirm that the user truly wants to lose their changes.

Environment
Internal and External

See Also
Refer to the example

Required License(s)
gateway

```
int UF_PART_reopen
(
    tag_t part,
    int scope,
    int mode,
    tag_t * new_part
)
```

tag_t	**part**	Input	Tag of part to reopen
int	**scope**	Input	0 = Only specified part 1 = Part and all subassemblies
int	**mode**	Input	0 = Ask confirmation if part is modified 1 = Reopen even if modified in session 2 = Reopen part(s) only if NOT modified in session
tag_t *	**new_part**	Output	This is different from part if reopening a part that is not used in an assembly.

UF_PART_save (view source)

Defined in: uf_part.h

Overview
Save the current work part and all its modified children (if an assembly) to disk.

Environment
Internal and External

Required License(s)
gateway

```
int UF_PART_save
(
    void
)
```

UF_PART_save_all (view source)

Defined in: uf_part.h

Overview

Saves all parts in session to disk with their original names. If any parts have filing errors while saving, count of errors is output, and an array of their part tags and an array of the associated error codes is allocated. The allocated arrays must be freed.

Return

Returned value indicates major error status as follows:
0 = No major error, but still check <count> for filing errors,
not 0 = Error code

Environment

Internal and External

Required License(s)

gateway

```
int UF_PART_save_all
(
    int * count,
    tag_t * * part_list,
    int * * error_list
)
```

int *	**count**	Output	Count of errors
tag_t * *	**part_list**	Output to UF_*free*	An allocated array containing the tags that failed to save. This must be freed by calling UF_free.
int * *	**error_list**	Output to UF_*free*	An allocated array of error codes corresponding to the part tags in part_list. This must be freed by calling UF_free.

UF_PART_save_as (view source)

Defined in: uf_part.h

Overview

Saves the work part to disk with a new part name. The name of the part in the current NX session is changed to the new name.

Note: This will save the current work part and all modified children (if an assembly) to the disk.
In Teamcenter Integration For NX, this can be used to create either a new revision, or a new item. This should not be used to create a new revision of a different item that already exists in the database.

Environment

Internal and External

Required License(s)

gateway

int UF_PART_save_as
(
const char * new_part_name
)

const char *	**new_part_name**	Input	This is the name of the new part you wish to save.

UF_PART_save_work_only (view source)

Defined in: uf_part.h

Overview
Save the current work part only to disk. Will not save any modified children (if an assembly).

Environment
Internal and External

Required License(s)
gateway

int UF_PART_save_work_only
(
void
)

UF_PART_set_compression_flags (view source)

Defined in: uf_part.h

Overview
Sets the compression flag of the part file. If the compression flag is set to true and you save the part, then the part is saved in compressed format.

Environment
Internal and External

See Also
UF_PART_ask_compression_flags
UF_PART_compress_flags_p_t
Refer to the example

Required License(s)
gateway

int UF_PART_set_compression_flags
(

```
    tag_t part,
    UF_PART_compress_flags_p_t compress_mask
)
```

tag_t	**part**	Input	Part to ask about compression flag
UF_PART_compress_flags_p_t	**compress_mask**	Input	bit field of compression flags settings

---

UF_PART_set_customer_area (view source)

Defined in: uf_part.h

Overview
Sets the customer area contents to the user defined specified string. The string should not be more than 132 bytes in length.

Return
Return code:
0 = No error
not 0 = Error code for invalid string.

Environment
Internal and External

See Also
UF_PART_ask_customer_area

Required License(s)
gateway

```
int UF_PART_set_customer_area
(
    tag_t part_tag,
    const char * customer_area
)
```

tag_t	**part_tag**	Input	Tag of the part in which to set the customer area.
const char *	**customer_area**	Input	Pointer to string to use as customer area contents. The maximum allowable length of the string is currently limited to 132 bytes.

UF_PART_set_description (view source)

Defined in: uf_part.h

Overview
Sets the description area contents to the user defined specified string. The string should not be more than 132 bytes in length.

Return

Return code:
0 = No error
not 0 = Error code for invalid string.

Environment
Internal and External

See Also
UF_PART_ask_description

Required License(s)
gateway

int UF_PART_set_description
(
tag_t part_tag,
const char * description
)

tag_t	**part_tag**	Input	Tag of the part in which to set the description area.
const char *	**description**	Input	Pointer to string to use as description area contents. The maximum allowable length of the string is currently limited to 132 bytes.

UF_PART_set_display_part (view source)

Defined in: uf_part.h

Overview
Sets the current display part. When working with an assembly, the "MAINTAIN_WORK_PART" option passed into UF_ASSEM_set_assem_options determines the actions of setting the work part. If NULL_TAG is passed in, the display part is set to the current work part.

Return
Error code:
0 = Success
1 = Part is not fully loaded

Environment
Internal and External

Required License(s)
gateway

int UF_PART_set_display_part
(
tag_t part
)

tag_t	**part**	Input	Tag of part to use for current display

UF_PART_set_enforce_piece_part (view source)

Defined in: uf_part.h

Overview
Sets or clears the enforce piece part flag of a part

Environment
Internal and External

Required License(s)
gateway

```
int UF_PART_set_enforce_piece_part
(
    tag_t part_tag,
    const logical status
)
```

tag_t	**part_tag**	Input	The tag of the part
const logical	**status**	Input	The enforced piece part status of the part. True will make the part an enforced piece part. False, will clear the enforced piece part status. It is an error to attempt to make an assembly part an enforced piece part.

UF_PART_set_fam_inst_save_dir (view source)

Defined in: uf_part.h

Overview
Sets the current part family instance save directory. When newly created part family instances are saved, they are placed in this directory.

Environment
Internal and External

History
This function was originally released in V15.0.

Required License(s)
gateway

```
int UF_PART_set_fam_inst_save_dir
(
    char * part_directory
)
```

char *	**part_directory**	Input	Directory Specification

UF_PART_set_family_save_dir (view source)

Defined in: uf_part.h

Overview
Sets the directory in which all members of the family are saved. It is stored with the family, so when family members are created on demand they still know where to be saved. This directory, when set, overrides the session default controlled by UF_PART_set_fam_inst_save_dir.

Environment
Internal and External

See Also
UF_PART_ask_family_save_dir

Required License(s)
gateway

```
int UF_PART_set_family_save_dir
(
    tag_t family,
    char * dir
)
```

tag_t	**family**	Input	Tag of the family to set the default save directory for
char *	**dir**	Input	Directory

UF_PART_set_status (view source)

Defined in: uf_part.h

Overview
Sets the integer that represents the status of the part. This status value is in the part file's header.

Environment
Internal and External

See Also
UF_PART_ask_status

Required License(s)
gateway

```
int UF_PART_set_status
(
    tag_t part_tag,
    const int status
)
```

tag_t	**part_tag**	Input	Tag of the part
const int	**status**	Input	Status of the part.

UF_PART_update_family_instance (view source)

Defined in: uf_part.h

Overview

Recreates and saves a part family instance if it can be found (using the current load options), and the instance is outdated with respect to its template part. If force_update is set to TRUE, the instance found is always updated. Instances are saved in the current family instance save directory, if they cannot be saved where they were found.

Environment

Internal and External

See Also

UF_PART_create_family_instance

Required License(s)

solid_modeling

```
int UF_PART_update_family_instance
(
    tag_t family,
    int member_index,
    logical force_update,
    tag_t * part,
    logical * saved,
    int * count,
    tag_t * * part_list,
    int * * error_list,
    char * * info
)
```

tag_t	**family**	Input	Object identifier of the family.
int	**member_index**	Input	Index of the family member.
logical	**force_update**	Input	Always update instance if found?
tag_t *	**part**	Output	Tag of the part containing the family instance.
logical *	**saved**	Output	Was the instance recreated and saved?
int *	**count**	Output	Count of errors.

tag_t * *	**part_list**	Output to UF_*free*	An allocated array containing the tags that failed to save. Use UF_free to deallocate memory when done.
int * *	**error_list**	Output to UF_*free*	An allocated array of error codes corresponding to the part tags in part_list. Use UF_free to deallocate memory when done.
char * *	**info**	Output to UF_*free*	Additional information about the instance. Use UF_free to deallocate memory when done.

UF_PART_update_jt_brep (view source)

Defined in: uf_part.h

Overview

Given a part that originated from JT data, update it with the BREP data.

NOTE: it's an error if the input part tag does not come from a JT file.

Return

Return code:
0 = Success
> 0 = Error code, use UF_get_fail_message to obtain error message string

Environment

Internal and External

History

Released in NX4

Required License(s)

gateway

```
int UF_PART_update_jt_brep
(
    const tag_t part_tag
)
```

const tag_t	**part_tag**	Input	The tag of the part you want to update the BREP of

---

UF_PART_update_jt_facets (view source)

Defined in: uf_part.h

Overview

Given a part that originated from JT data, update the facet data to the latest information in the JT file.

NOTE: it's an error if the input part tag does not come from a JT file.

Return
Return code:
0 = Success
> 0 = Error code, use UF_get_fail_message to obtain error message string

Environment
Internal and External

History
Released in NX4

Required License(s)
gateway

int UF_PART_update_jt_facets
(
const tag_t part_tag
)

const tag_t	**part_tag**	Input	The tag of the part you want to update the BREP of