uc5027 (view source)

Defined in: uf_obj.h

Overview
uc5027 read object name -- replaced by UF_OBJ_ask_name

Required License(s)
gateway

```
int uc5027
(
    tag_t np1,
    char cr2 [ UF_OBJ_NAME_BUFSIZE ] ,
    int * ir3
)
```

tag_t	**np1**	Output
char	**cr2 [UF_OBJ_NAME_BUFSIZE]**	
int *	**ir3**	

---

uc5028 (view source)

Defined in: uf_obj.h

Overview
uc5028 find named object -- replaced by UF_OBJ_cycle_by_name

Required License(s)
gateway

```
int uc5028
(
    const char * cp1,
    int ip2,
    tag_t * nr3
)
```

const char *	**cp1**	Input
int	**ip2**	Input
tag_t *	**nr3**	

---

uc5029 (view source)

Defined in: uf_obj.h

Overview
uc5029 add/modify object name -- replaced by UF_OBJ_set_name

Required License(s)
gateway

```
int uc5029
(
    tag_t np1,
    const char * cp2,
    int ip3
)
```

tag_t	**np1**	Input
const char *	**cp2**	
int	**ip3**	

---

uc502a (view source)

Defined in: uf_obj.h

Overview
uc502a read object font -- NO REPLACEMENT (use UF_OBJ_ask_display_properties)

Required License(s)
gateway

```
int uc502a
(
    tag_t np1,
    char cr2 [ 31 ]
)
```

tag_t	**np1**	Output
char	**cr2 [31]**	

---

uc502e (view source)

Defined in: uf_obj.h

Overview
uc502e temporarily modify object defaults -- NO REPLACEMENT (use default creation routines above)

Required License(s)
gateway

int uc502e
(
int ip1,
int ip2,
int ip3,
int ip4,
const char * cp5
)

int	**ip1**	
int	**ip2**	
int	**ip3**	
int	**ip4**	Input
const char *	**cp5**	

---

uc502m (view source)

Defined in: uf_obj.h

Overview
uc502m edit object font -- NO REPLACEMENT (use UF_OBJ_set_font)

Required License(s)
gateway

int uc502m
(
tag_t np1,
const char * cp2
)

tag_t	**np1**	Input
const char *	**cp2**	

---

uc502n (view source)

Defined in: uf_obj.h

Overview
uc502n read part file data and set default object -- NO REPLACEMENT (use default creation routines

above)

Required License(s)

gateway

int uc502n
(
int * ia1,
int * ia2,
int * ia3,
char ca4 [UF_OBJ_NAME_BUFSIZE]
)

int *	**ia1**	
int *	**ia2**	
int *	**ia3**	Input / Output
char	**ca4 [UF_OBJ_NAME_BUFSIZE]**	

---

uc502r (view source)

Defined in: uf_obj.h

Overview

uc502r read current font definition file name -- NO REPLACEMENT

Required License(s)

gateway

int uc502r
(
int ip1,
char ca2 [MAX_FSPEC_BUFSIZE]
)

int	**ip1**	Input / Output
char	**ca2 [MAX_FSPEC_BUFSIZE]**	

---

uc5203 (view source)

Defined in: uf_obj.h

Overview

uc5203 cycle by object name -- replaced by UF_OBJ_cycle_by_name

Required License(s)

gateway

void uc5203
(
tag_t * na1,
const char * cp2,
int ip3
)

tag_t *	**na1**	Input
const char *	**cp2**	
int	**ip3**	

---

uf5025 (view source)

Defined in: uf_obj.h

Overview

uf5025 read part file data and set default object -- replaced by default creation routines above

Required License(s)

gateway

void uf5025
(
int * ia1,
int * ia2,
int * ia3,
int * ia4
)

int *	**ia1**
int *	**ia2**
int *	**ia3**
int *	**ia4**

---

uf5026 (view source)

Defined in: uf_obj.h

Overview

uf5026 temporarily modify object defaults -- replaced by default creation routines above

Required License(s)
gateway

```
void uf5026
(
    const int * ip1,
    const int * ip2,
    const int * ip3,
    const int * ip4,
    const int * ip5
)
```

const int *	**ip1**
const int *	**ip2**
const int *	**ip3**
const int *	**ip4**
const int *	**ip5**

UF_OBJ_ask_cre_mod_versions (view source)

Defined in: uf_obj.h

Overview
Returns the version of the part in which the given object was created and the version in which it was last modified. The version numbers returned are from 1 to the highest version number of the part, unless there is an error, in which case the version numbers are set to 0. The UF_PART_<xxxx> routines which deal with part save histories can be used to obtain information about the versions of a part.

Environment
Internal and External

See Also
UF_PART_ask_nth_history
UF_PART_ask_num_histories
UF_PART_ask_part_history
UF_PART_clear_history_list
UF_PART_create_history_list
UF_PART_delete_history_list

For example please refer to the example

Required License(s)
gateway

```
int UF_OBJ_ask_cre_mod_versions
(
    tag_t object,
    int * creation_version,
```

```
    int * lastmod_version
)
```

tag_t	**object**	Input	Object Identifier
int *	**creation_version**	Output	Object's creation version
int *	**lastmod_version**	Output	Object's last modification version

---

UF_OBJ_ask_cre_settings (view source)

Defined in: uf_obj.h

Overview

Reads the work part's color, line font, and width object creation settings for the type and subtype specified. If the type is an invalid object type or if no object creation settings exist for the type and subtype combination, then an error is returned. This routine uses the settings structure UF_OBJ_cre_settings_s .

Environment

Internal and External

See Also

UF_OBJ_cre_settings_t
For information on type, subtype and properties click here.

Required License(s)

gateway

```
int UF_OBJ_ask_cre_settings
(
    const int type,
    const int subtype,
    const int property,
    UF_OBJ_cre_settings_p_t settings
)
```

const int	**type**	Input	type to ask settings of
const int	**subtype**	Input	Subtype of type to ask settings of
const int	**property**	Input	Property of type/subtype to ask settings of
UF_OBJ_cre_settings_p_t	**settings**	Output	Color, line font and width object creation settings for the specified type and subtype.

UF_OBJ_ask_def_cre_settings (view source)

Defined in: uf_obj.h

Overview
Reads the work part's default color, line font, and width object creation settings. This routine uses the settings structure UF_OBJ_cre_settings_s.

Environment
Internal and External

See Also
UF_OBJ_cre_settings_t

Required License(s)
gateway

int UF_OBJ_ask_def_cre_settings
(
UF_OBJ_cre_settings_p_t settings
)

UF_OBJ_cre_settings_p_t	**settings**	Output	Default color, line font and width object creation settings for the specified type and subtype.

UF_OBJ_ask_display_properties (view source)

Defined in: uf_obj.h

Overview
Returns the display properties (layer, color, blank status, line width, and font) for an object.

Environment
Internal and External

See Also
For structure see UF_OBJ_disp_props_t

Required License(s)
gateway

int UF_OBJ_ask_display_properties
(
tag_t object_id,
UF_OBJ_disp_props_p_t disp_props
)

tag_t	**object_id**	Input	Object identifier of object to query
UF_OBJ_disp_props_p_t	**disp_props**	Output	Display properties of object of inquiry

---

UF_OBJ_ask_extended_type_and_subtype (view source)

Defined in: uf_obj.h

Overview

Returns the extended object type of a tagged NX object. (extended object types have values over 256). The majority of these objects are listed in the uf_object_types.h header file.

In addition to the objects listed in uf_object_types.h, there are other NX objects uniquely identified by a tag. Among these objects are expressions and parts. These objects, which are not listed in the uf_object_types.h header file, produce the type UF_OBJ_EXPR_TYPE and UF_OBJ_PART_TYPE with a subtype of 0.

The types and subtypes this function returns for expressions and parts are not useful in any other Open API routines. These types are provided only to allow you to determine the class of a NX object through its identifier (i.e. its tag) which can assist you in determining the other Open API routines that you can use in conjunction with the object. For a more detailed discussion of classes of objects and the Open API routines available to these objects, please see the Open API Programmer's Guide, "The NX Object Model".

The difference between the type returned by UF_OBJ_ask_type_and_subtype and this routine is, UF_OBJ_ask_type_and_subtype returns UF_extended_type or UF_extended_displayable_type for
all the extended type objects except for some types like parts and expressions.
Whereas this routine returns the object's real type.

Environment

Internal and External

See Also

UF_OBJ_ask_type_and_subtype

Required License(s)

gateway

```
int UF_OBJ_ask_extended_type_and_subtype
(
    tag_t object_id,
    int * type,
    int * subtype
)
```

tag_t	**object_id**	Input	Object identifier of object to query
int *	**type**	Output	Type of object
int *	**subtype**	Output	Subtype of object

UF_OBJ_ask_face_analysis (view source)

Defined in: uf_obj.h

Overview

Asks the face analysis attribute of a face. If the attribute is TRUE, the face is displayed using analysis data in face analysis views. The parameters defining the details of the display are specified by UF_DISP_set_srfanl_params.

Environment

Internal

See Also

UF_DISP_set_srfanl_params

Required License(s)

gateway

int UF_OBJ_ask_face_analysis
(
tag_t face,
logical * srfanl
)

tag_t	**face**	Input	Object identifier
logical *	**srfanl**	Output	Face analysis data flag: TRUE = Face is displayed with face analysis data in face analysis views FALSE = off

---

UF_OBJ_ask_name (view source)

Defined in: uf_obj.h

Overview

Required License(s)

gateway

int UF_OBJ_ask_name
(
tag_t object_id,
char * name
)

tag_t	**object_id**	Input	Object identifier of object to inquire upon.
char *	**name**	Output	Name of object

UF_OBJ_ask_name_origin (view source)

Defined in: uf_obj.h

Overview
Returns the name origin of the specified object. If the object you query is not of an object type listed in uf_object_types.h or if the object has no name, an error code returns and the origin is set to (0,0,0).

Environment
Internal and External

See Also
UF_OBJ_set_name_origin

History
Originally released in V17.0

Required License(s)
gateway

```
int UF_OBJ_ask_name_origin
(
    tag_t object_id,
    double origin [ 3 ]
)
```

tag_t	**object_id**	Input	Object identifier of object to inquire upon.
double	**origin [3]**	Output	Origin in absolute coordinates where the name of the object will be displayed.

UF_OBJ_ask_owning_part (view source)

Defined in: uf_obj.h

Overview
Returns the tag of the part which contains the specified object_in_part.

Environment
Internal and External

Required License(s)
gateway

```
int UF_OBJ_ask_owning_part
(
    tag_t object_in_part,
    tag_t * part_tag
)
```

tag_t	**object_in_part**	Input	Tag of object for which the part is required.
tag_t *	**part_tag**	Output	part containing object_in_part

UF_OBJ_ask_partially_shaded (view source)

Defined in: uf_obj.h

Overview
Returns the partially shaded status for a single solid face.

Environment
Internal and External

Required License(s)
gateway

```
int UF_OBJ_ask_partially_shaded
(
    tag_t object,
    logical * shaded
)
```

tag_t	**object**	Input	Object identifier of solid face to read attribute from
logical *	**shaded**	Output	Shaded status of the solid face TRUE = shading is on FALSE = shading is off

UF_OBJ_ask_status (view source)

Defined in: uf_obj.h

Overview
Returns the objects status, UF_OBJ_DELETED, UF_OBJ_TEMPORARY, UF_OBJ_CONDEMNED,
and UF_OBJ_ALIVE. This function should only be used to get information on objects within the part, not the part itself.

NOTE: You are strongly advised to avoid doing anything to non-alive objects unless you are familiar with their use. NX may delete or reuse these objects at any time. Some of these objects do not get filed with the part.

Return
Returns 1 of 4 statuses
UF_OBJ_DELETED
UF_OBJ_TEMPORARY
UF_OBJ_CONDEMNED
UF_OBJ_ALIVE

Environment
Internal and External

Required License(s)
gateway

```
int UF_OBJ_ask_status
(
    tag_t object
)
```

tag_t	**object**	Input	The object for which you are asking status

---

UF_OBJ_ask_translucency (view source)

Defined in: uf_obj.h

Overview
Asks the translucency of faces, facets and bodies.

Environment
Internal and External

History
Introduced in NX 2.0.2

Required License(s)
gateway

```
int UF_OBJ_ask_translucency
(
    tag_t object,
    UF_OBJ_translucency_p_t translucency
)
```

tag_t	**object**	Input	The object for which you are asking the translucency It must be of one of the following types: UF_solid_type, with one of these subtypes: UF_solid_body_type UF_solid_face_type UF_faceted_model_type UF_component_type
UF_OBJ_translucency_p_t	**translucency**	Output	percent translucent 0 - 100

UF_OBJ_ask_type_and_subtype (view source)

Defined in: uf_obj.h

Overview
Returns the object type and subtype of a tagged NX object.
The majority of these objects are listed in the uf_object_types.h
header file.

In addition to the objects listed in uf_object_types.h, there are other

NX objects uniquely identified by a tag. Among these objects are expressions and parts. These objects, which are not listed in the uf_object_types.h header file, produce the type UF_OBJ_EXPR_TYPE and UF_OBJ_PART_TYPE with a subtype of 0.

The types and subtypes this function returns for expressions and parts are not useful in any other Open API routines. These types are provided only to allow you to determine the class of a NX object through its identifier (i.e. its tag) which can assist you in determining the other Open API routines that you can use in conjunction with the object. For a more detailed discussion of classes of objects and the Open API routines available to these objects, please see the Open API Programmer's Guide, "The NX Object Model".

Environment
Internal and External

Required License(s)
gateway

```
int UF_OBJ_ask_type_and_subtype
(
    tag_t object_id,
    int * type,
    int * subtype
)
```

tag_t	**object_id**	Input	Object identifier of object to query
int *	**type**	Output	Type of object
int *	**subtype**	Output	Subtype of object

UF_OBJ_cycle_all (view source)

Defined in: uf_obj.h

Overview
Returns all objects in a given part on all layers regardless of their current status or displayability. This includes temporary (system created) objects. It does not return expressions.

This function returns the tag of the object that was found. To continue cycling, pass the returned object in as the second argument to this function.

NOTE: You are strongly advised to avoid doing anything to non-alive objects unless you are familiar with their use. NX may delete or reuse these objects at any time. Some of these objects do not get filed with the part.

NOTE: This routine is invalid for partially loaded parts. If you call this function using a partially loaded part, it returns a NULL_TAG from the beginning of the cycle.

Do not attempt to delete objects when cycling the database in a loop. Problems

can occur when trying to read the next object when the current object has been deleted. To delete objects, save an array with the objects in it, and then when you have completed cycling, use UF_OBJ_delete_array_of_objects to delete the saved array of objects.

Return
This function returns the object identifier of the next object.
The cycle is completed when a NULL_TAG is returned

Environment
Internal and External

See Also
Please refer to the usage and the example

Required License(s)
gateway

tag_t UF_OBJ_cycle_all
(
tag_t part_tag,
tag_t object
)

tag_t	**part_tag**	Input	Tag of part you wish to cycle
tag_t	**object**	Input	Begin the cycle by passing in object = NULL_TAG

UF_OBJ_cycle_by_name (view source)

Defined in: uf_obj.h

Overview
Cycle the objects of a given name in current work part. Note that the name of an object of type UF_occ_instance_type is not necessarily the instance name.

Note that the name must match exactly. If the name of an object is "abc ", and the name passed here is "abc", then the object will not be found because of the trailing space.

When using UF_OBJ_cycle_by_name, features are only returned if they are the only object with that name. If there is any possibility that there could be objects with the same name (that are not features), then call UF_OBJ_cycle_by_by_name_and_type using UF_feature_type in order to find the named features.
Please refer to the example

Do not attempt to delete objects when cycling the database in a loop. Problems can occur when trying to read the next object when the current object has been deleted. To delete objects, save an array with the objects in it, and then when you have completed cycling, use UF_OBJ_delete_array_of_objects to delete the saved array of objects.

Environment
Internal and External

Required License(s)

gateway

int UF_OBJ_cycle_by_name
(
const char * name,
tag_t * object
)

const char *	**name**	Input	Name of objects on which to cycle
tag_t *	**object**	Input / Output	On input the object found by the last call to this routine. Begin the cycle by passing in object = NULL_TAG On output, the next object of the given name. Outputs a NULL_TAG when the cycle is finished.

UF_OBJ_cycle_by_name_and_type (view source)

Defined in: uf_obj.h

Overview

Cycles objects in the specified part by name and object type. Note that the name of an object of type UF_occ_instance_type is not necessarily the instance name.

Do not attempt to delete objects when cycling the database in a loop. Problems can occur when trying to read the next object when the current object has been deleted. To delete objects, save an array with the objects in it, and then when you have completed cycling, use UF_OBJ_delete_array_of_objects to delete the saved array of objects.

Environment

Internal and External

History

This function was originally released in V15.0.

Required License(s)

gateway

int UF_OBJ_cycle_by_name_and_type
(
tag_t part_tag,
const char * name,
int type,
logical use_occ,
tag_t * object
)

tag_t	**part_tag**	Input	Tag of part in which to cycle.
const char *	**name**	Input	Name of objects on which to cycle.
int	**type**	Input	Object type on which to cycle (see uf_object_types.h)

logical	**use_occ**	Input	Occurrence filter: TRUE = include occurrences FALSE = exclude occurrrences
tag_t *	**object**	Input / Output	On input the object found by the last call to this routine. Begin the cycle by passing in object = NULL_TAG On output, the next object of the given name. Outputs a NULL_TAG when the cycle is finished.

UF_OBJ_cycle_by_name_and_type_extended (view source)

Defined in: uf_obj.h

Overview

Cycles objects in the specified part by name and object type. Note that the name of an object of type UF_occ_instance_type is not necessarily the instance name.

This works like UF_OBJ_cycle_by_name_and_type.
The difference between this routine and UF_OBJ_cycle_by_name_and_type is that UF_OBJ_cycle_by_name_and_type returns all the extended objects (extended object types have values over 256) if either UF_extended_type or UF_extended_displayable_type is passed in as type. Whereas this routine does not return any objects for those two types

Do not attempt to delete objects when cycling the database in a loop. Problems can occur when trying to read the next object when the current object has been deleted. To delete objects, save an array with the objects in it, and then when you have completed cycling, use UF_OBJ_delete_array_of_objects to delete the saved array of objects.

Environment

Internal and External

See Also

UF_OBJ_cycle_by_name_and_type

History

This function was originally released in V15.0.

Required License(s)

gateway

```
int UF_OBJ_cycle_by_name_and_type_extended
(
    tag_t part_tag,
    const char * name,
    int type,
    logical use_occ,
    tag_t * object
)
```

tag_t	**part_tag**	Input	Tag of part in which to cycle.
const char *	**name**	Input	Name of objects on which to cycle.
int	**type**	Input	Object type on which to cycle (see uf_object_types.h)

logical	**use_occ**	Input	Occurrence filter: TRUE = include occurrences FALSE = exclude occurrrences
tag_t *	**object**	Input / Output	On input the object found by the last call to this routine. Begin the cycle by passing in object = NULL_TAG On output, the next object of the given name. Outputs a NULL_TAG when the cycle is finished.

UF_OBJ_cycle_objs_in_part (view source)

Defined in: uf_obj.h

Overview

Returns all objects in the part of the specified types on all layers (including layers not writable or viewable) regardless of their current state of displayability. This routine does not return expressions, condemned objects, temporary (system created) objects, or sleepy objects. An object that is asleep is one that has been removed from the model. For example, if you blend an edge, then the edge is asleep. The WCS will not be returned unless it has been saved.

NOTE: This routine cycles the features in a part when the type specified is UF_feature_type.

Do not attempt to delete objects when cycling the database in a loop. Problems can occur when trying to read the next object when the current object has been deleted. To delete objects, save an array with the objects in it, and then when you have completed cycling, use UF_OBJ_delete_array_of_objects to delete the saved array of objects.

Environment

Internal and External

See Also

For example please refer to the example

Required License(s)

gateway

```
int UF_OBJ_cycle_objs_in_part
(
    tag_t part_tag,
    int type,
    tag_t * object
)
```

tag_t	**part_tag**	Input	Tag of part you wish to cycle
int	**type**	Input	Type of object on which to cycle
tag_t *	**object**	Input / Output	On input the object found by the last call to this routine. if this routine has not been called yet, then set object=NULL_TAG to start cycling. On output the next object of the type specified. If there is no object, and the cycling is complete a NULL_TAG is returned.

UF_OBJ_cycle_objs_in_part1 (view source)

Defined in: uf_obj.h

Overview

Returns all objects in the part of the specified types that are found on layers 1 thru 256 regardless of their current state of displayability. This routine does not return expressions, condemned objects, temporary (system created) objects, or sleepy objects. An object that is asleep is one that has been removed from the model.
For example, if you blend an edge, then the edge is asleep. The WCS will not be returned unless it has been saved.

NOTE: This routine cycles the features in a part when the type specified is UF_feature_type.

Do not attempt to delete objects when cycling the database in a loop. Problems can occur when trying to read the next object when the current object has been deleted. To delete objects, save an array with the objects in it, and then when you have completed cycling, use UF_OBJ_delete_array_of_objects to delete the saved array of objects.

Environment

Internal and External

See Also

For example please refer to the example

History

NX10.0.0

Required License(s)

gateway

```
int UF_OBJ_cycle_objs_in_part1
(
    tag_t part_tag,
    int type,
    tag_t * object
)
```

tag_t	**part_tag**	Input	Tag of part you wish to cycle
int	**type**	Input	Type of object on which to cycle
tag_t *	**object**	Input / Output	On input the object found by the last call to this routine. if this routine has not been called yet, then set object=NULL_TAG to start cycling. On output the next object of the type specified. If there is no object, and the cycling is complete a NULL_TAG is returned.

UF_OBJ_cycle_typed_objs_in_part (view source)

Defined in: uf_obj.h

Overview

This routine has been added to deal with extended object types (extended object types have values over 256).

Returns all objects in the part of the specified types on all layers regardless of their current state of displayability. This routine does not return expressions, condemned objects, temporary (system created) objects, or sleepy objects. An object that is asleep is one that has been removed from the model. For example, if you blend an edge, then the edge is asleep. The WCS will not be returned unless it has been saved.

Although this works like UF_OBJ_cycle_objs_in_part, there is a difference. When type is UF_extended_type, this routine (UF_OBJ_cycle_typed_objs_in_part) will not returns any objects. Whereas UF_OBJ_cycle_objs_in_part will return all the objects with extended entity types for type UF_extended_type.

For more details on how UF_OBJ_cycle_objs_in_part works, please see comment for UF_OBJ_cycle_objs_in_part

NOTE: This routine cycles the features in a part when the type specified is UF_feature_type.

Do not attempt to delete objects when cycling the database in a loop. Problems can occur when trying to read the next object when the current object has been deleted. To delete objects, save an array with the objects in it, and then when you have completed cycling, use UF_OBJ_delete_array_of_objects to delete the saved array of objects.

See Also

UF_OBJ_cycle_objs_in_part

Required License(s)

gateway

```
int UF_OBJ_cycle_typed_objs_in_part
(
    tag_t part_tag,
    int type,
    tag_t * object
)
```

tag_t	**part_tag**	Input	Tag of part you wish to cycle
int	**type**	Input	Type of object on which to cycle
tag_t *	**object**	Input / Output	On input the object found by the last call to this routine. if this routine has not been called yet, then set object=NULL_TAG to start cycling. On output the next object of the type specified. If there is no object, and the cycling is complete a NULL_TAG is returned.

UF_OBJ_delete_array_of_objects (view source)

Defined in: uf_obj.h

Overview

Deletes an array of objects. This is considerably faster than deleting the objects one at a time. This routine should always be used when more than one object is to be deleted.

Do not attempt to delete objects while cycling the database in a looping structure. Problems can occur when trying to read the next object and delete objects in a looping structure. To delete objects, we recommend that you:

Save the objects in an array and then perform the delete function after you have completed the cycling by calling UF_OBJ_delete_array_of_objects.

A return value of zero does not guarantee that all of the objects were deleted. To determine if an particular object was deleted, use UF_OBJ_ask_status. If statuses is NULL, then the return code from this function will be 0 if the return code from UF_OBJ_delete_object would have been zero for all the objects. A non-zero status can either mean that an individual object could not be deleted, or that some other fatal error occurred. If statuses is not null, then the individual delete status for each object is returned in the statuses array. The return code from the function will be non-zero if some more global failure occurred.

Environment

Internal and External

See Also

UF_OBJ_delete_object

History

Originally released in V16.0

Required License(s)

gateway

```
int UF_OBJ_delete_array_of_objects
(
    int num_objects,
    tag_t object_id [ ] ,
    int * * statuses
)
```

int	**num_objects**	Input	The count of the objects in the object_id array.
tag_t	**object_id [ ]**	Input	Array of object identifiers to be deleted.
int * *	**statuses**	Input / Output to UF_*free*	This should be the address of pointer. If it is NULL, then individual status for the delete operation on each object will not be given. If it is non-NULL, then an array will be allocated, and the individual status for each object placed in this array. The status for individual objects can be: 0 - object was passed on to the delete algorithm UF_OBJ_object_can_not_be_deleted UF_OBJ_err_bad_parameter_number_1 This status will be the same as what would have been returned had the single object been passed to UF_OBJ_delete_object. A value of 0 does not necessarily mean that the object was

			deleted, this can only be determined by calling UF_OBJ_ask_status. If a non-NULL pointer is passed in, then the user must free this returned status array by using UF_free.

UF_OBJ_delete_name (view source)

Defined in: uf_obj.h

Overview

Deletes the name from the specified object. Only objects with an object type listed in uf_object_types.h can have their names deleted.

Environment

Internal and External

Required License(s)

gateway

int UF_OBJ_delete_name
(
tag_t object_id
)

tag_t	**object_id**	Input	Object identifier of object whose name is to be deleted.

UF_OBJ_delete_object (view source)

Defined in: uf_obj.h

Overview

Deletes an object. A return value of zero does not guarantee that an object is deleted. To determine if an object is deleted, use UF_OBJ_ask_status.

It is not allowed to delete the current WCS, if you try, an error will be returned.

Do not attempt to delete objects while cycling the database in a looping structure. Problems can occur when trying to read the next object and delete objects in a looping structure. To delete objects, we recommend:

Save the objects in an array and then perform the delete function using UF_OBJ_delete_array_of_objects after you have completed the cycling.

If you have multiple objects to delete, you should use the routine UF_OBJECT_delete_array_of_objects, as it will be much faster.

Note: If a type-specific function exists for the type of object being deleted, please use that type-specific function instead of this general use function. For example, to delete a view use UF_VIEW_delete, to delete a drawing use UF_DRAW_delete_drawing, etc.

Environment
Internal and External

See Also
UF_OBJ_ask_status
UF_OBJ_delete_array_of_objects

Required License(s)
gateway

int UF_OBJ_delete_object
(
tag_t object_id
)

tag_t	**object_id**	Input	Object identifier of object to delete

---

UF_OBJ_is_def_cre_color (view source)

Defined in: uf_obj.h

Overview
Query if the work part's type/subtype color setting is using the default color setting.

Environment
Internal and External

See Also
For information on type, subtype and properties click here.

Required License(s)
gateway

int UF_OBJ_is_def_cre_color
(
const int type,
const int subtype,
const int property,
logical * is_default
)

const int	**type**	Input	Type to query
const int	**subtype**	Input	Subtype to query
const int	**property**	Input	Property of type/subtype to query
logical *	**is_default**	Output	TRUE = Color is set to default FALSE = Color is not set to default

---

UF_OBJ_is_def_cre_line_font (view source)

Defined in: uf_obj.h

Overview
Query if the work part's type/subtype line font setting is using the default line font setting.

Environment
Internal and External

See Also
For information on type, subtype and properties click here.

Required License(s)
gateway

```
int UF_OBJ_is_def_cre_line_font
(
    const int type,
    const int subtype,
    const int property,
    logical * is_default
)
```

const int	**type**	Input	Type to query
const int	**subtype**	Input	Subtype to query
const int	**property**	Input	Property of type/subtype to query
logical *	**is_default**	Output	TRUE = Line font is set to default FALSE = Line font is not set to default

---

UF_OBJ_is_def_cre_width (view source)

Defined in: uf_obj.h

Overview
Query if the work part's type/subtype line width setting is using the default line width setting.

Environment
Internal and External

See Also
For information on type, subtype and properties click here.

Required License(s)
gateway

```
int UF_OBJ_is_def_cre_width
(
    const int type,
    const int subtype,
    const int property,
    logical * is_default
)
```

const int	**type**	Input	Type to query
const int	**subtype**	Input	Subtype to query
const int	**property**	Input	Property of type/subtype to query
logical *	**is_default**	Output	TRUE = Line width is set to default FALSE = Line width is not set to default

---

UF_OBJ_is_displayable (view source)

Defined in: uf_obj.h

Overview

Returns a logical indicating whether tagged NX object is displayable, and therefore possesses attributes of color, width, font, and layer.

Although an object possesses these displayable attributes, it does not mean that the object is ever actually displayed in the graphics window in an interactive NX session. Some objects possess these characteristics for purposes internal to NX.

Return

Returns a logical which indicates whether object is displayable

Environment

Internal and External

Required License(s)

gateway

```
int UF_OBJ_is_displayable
(
    tag_t object_id,
    logical * is_displayable
)
```

tag_t	**object_id**	Input	Object identifier of object to query
logical *	**is_displayable**	Output	TRUE = This object is displayable FALSE = This object is not displayable

---

UF_OBJ_is_object_a_promotion (view source)

Defined in: uf_obj.h

Overview
Queries if an object is a promoted object. Returns TRUE if object is a promotion.

Environment
Internal and External

Required License(s)
gateway

```
logical UF_OBJ_is_object_a_promotion
(
    tag_t object
)
```

tag_t	**object**	Input	Tag of object to query

UF_OBJ_is_transferable (view source)

Defined in: uf_obj.h

Overview
Determines whether the given tagged NX object is a transferable or not. You can potentially export a transferable object using UF_PART_export or UF_PART_export_with_options, or transfer to a new component using UF_ASSEM_create_component_part.

Although a particular object may return as being transferable, there is no guarantee that the objects of that type will be transferred. For example, if an object of a transferable type depends upon an object which is not transferable, then the original object will not be transferred.

Environment
Internal and External

See Also
UF_PART_export
UF_PART_export_with_options
UF_ASSEM_create_component_part
Please refer to the example

Required License(s)
gateway

```
int UF_OBJ_is_transferable
(
    tag_t object_id,
    logical * is_transferable
)
```

tag_t	**object_id**	Input	Object identifier of object to query
logical *	**is_transferable**	Output	TRUE = This object is transferable FALSE = This object is not transferable

---

UF_OBJ_modify_defaults (view source)

Defined in: uf_obj.h

Overview

UF_OBJ_modify_defaults -- replaced by default routines above

Required License(s)

gateway

```
int UF_OBJ_modify_defaults
(
    int default_color,
    int default_layer,
    int default_width,
    int default_font
)
```

int	**default_color**	Input
int	**default_layer**	Input
int	**default_width**	Input
int	**default_font**	Input

UF_OBJ_replace_object_array_data (view source)

Defined in: uf_obj.h

Overview

Replaces the geometry of the n'th element of the target object array with
the geometry of the n'th element of the source object array. Neither object
can be an occurrence but both objects may belong to the same component part
in an assembly. Both objects must be alive (see UF_OBJ_ask_status)
and both objects must be distinct and belong to the same part.

The target objects get added to the update list and any references to
these objects are maintained.
The source objects get added to the delete list.
Users must call UF_MODL_update() at the appropriate time to update
the target objects, delete the source objects and
propagate the changes to the model.

The type and subtype of each pair of objects must be valid for the routine
but they are not required to be the same. It is expected that the source

object is a newly created object with no extra user defined attributes attached, and that it is not referenced by other objects.

Only valid geometry can exist in the objects, which consists of the following (all subtypes are valid):
UF_point_type
UF_line_type
UF_circle_type
UF_conic_type
UF_spline_type

The display details [layer, color, line font and width, blank and highlight status] and all user defined attributes, including names of the target objects are all maintained. Links and references to the target objects are preserved.

The reference set of the source objects are copied to the respective target objects. If a source object is of a different type than the corresponding target object, any occurrences existing based on the target object are deleted and recreated by this routine.

Environment
Internal and External

See Also
UF_OBJ_replace_object_data

Required License(s)
gateway

```
int UF_OBJ_replace_object_array_data
(
    int num_objects,
    tag_t * target_objects,
    tag_t * source_objects
)
```

int	**num_objects**	Input	Number of objects whose geometry is to be replaced
tag_t *	**target_objects**	Input	Target objects whose geometry is to be replaced
tag_t *	**source_objects**	Input	Source objects whose geometry is to be used as replacements for target_objects

---

UF_OBJ_replace_object_data (view source)

Defined in: uf_obj.h

Overview
Replaces the geometry of the target object with the geometry of the source object. Neither object can be an occurrence but both objects may belong to the same component part in an assembly. Both objects must be alive (see UF_OBJ_ask_status) and both objects must be distinct and belong to the same part.

The target object gets updated and any references to this object are maintained. The source object gets deleted.

The type and subtype of each object must be valid for the routine but they are not required to be the same. It is expected that the source object is a newly created object with no extra user defined attributes attached, and that it is not referenced by other objects.

Only valid geometry can exist in either object which consists of the following (all subtypes are valid):
UF_point_type
UF_line_type
UF_circle_type
UF_conic_type
UF_spline_type

The display details [layer, color, line font and width, blank and highlight status] and all user defined attributes, including name of the target object are all maintained. Links and references to the target object are preserved. The reference set of the source object is copied to the target object. If the source object is a different type than the target object, any occurrences existing based on the target object are deleted and recreated by this routine.

NOTE: This routine will update the model. If this is not desirable, see UF_OBJ_replace_object_array_data()

Environment
Internal and External

See Also
UF_OBJ_replace_object_array_data

Required License(s)
gateway

```
int UF_OBJ_replace_object_data
(
    tag_t orig_obj,
    tag_t new_obj
)
```

tag_t	**orig_obj**	Input	Target object whose geometry is to be replaced.
tag_t	**new_obj**	Input	Source object whose geometry is to be used as a replacement for orig_obj.

UF_OBJ_reset_defaults (view source)

Defined in: uf_obj.h

Overview
UF_OBJ_reset_defaults -- replaced by default routines above

Required License(s)
gateway

```
int UF_OBJ_reset_defaults
(
    int default_color,
    int default_layer,
    int default_width,
    int default_font
)
```

int	**default_color**	Input
int	**default_layer**	Input
int	**default_width**	Input
int	**default_font**	Input

UF_OBJ_return_prev_defaults (view source)

Defined in: uf_obj.h

Overview
UF_OBJ_return_prev_defaults -- replaced by default routines above

Required License(s)
gateway

```
int UF_OBJ_return_prev_defaults
(
    void
)
```

UF_OBJ_reverse_blank_all (view source)

Defined in: uf_obj.h

Overview
Reverses the blank status of all displayable objects on selectable layers. Uses the work view layer mask.

Environment
Internal and External

Required License(s)
gateway

```
int UF_OBJ_reverse_blank_all
(
    void
)
```

UF_OBJ_set_blank_status (view source)

Defined in: uf_obj.h

Overview
Modifies the blank status of an object. Use UF_OBJ_ask_display_properties to determine the object's current blank status.

If the object is a sketch or a group, all members of the sketch or group will have their blank status modified. For groups, this includes all members of any sub-groups.

Environment
Internal and External

See Also
UF_OBJ_ask_display_properties

History
V18.0 Comments about sketches and groups added

Required License(s)
gateway

```
int UF_OBJ_set_blank_status
(
    tag_t object_id,
    int blank_status
)
```

tag_t	**object_id**	Input	Object identifier of object whose blank status is to be changed.
int	**blank_status**	Input	Blank status to assign. Use one of the following string defined constants: UF_OBJ_NOT_BLANKED UF_OBJ_BLANKED

UF_OBJ_set_color (view source)

Defined in: uf_obj.h

Overview
Modifies the color of an object. Use UF_OBJ_ask_display_properties to determine the object's current color.

Note: There must be a work part present before calling this function.

Environment
Internal and External

See Also

See the string defined color constants in the types and symbols .
UF_OBJ_ask_display_properties

Required License(s)
gateway

int UF_OBJ_set_color
(
tag_t object_id,
int color
)

tag_t	**object_id**	Input	Object identifier of object whose color is to be changed.
int	**color**	Input	Color to assign. Must be one of the following values [1-216]

UF_OBJ_set_cre_color (view source)

Defined in: uf_obj.h

Overview
Set the specified type and subtype's color object creation setting in the work part. If the type is not a valid object type or if no object creation settings exist for the type and subtype combination then an error is returned.

Environment
Internal and External

See Also
See the string defined color constants in the types and symbols
For information on type, subtype and properties click here.

Required License(s)
gateway

int UF_OBJ_set_cre_color
(
const int type,
const int subtype,
const int property,
const int color
)

const int	**type**	Input	Type to set color of
const int	**subtype**	Input	Subtype to set color of
const int	**property**	Input	Property of type/subtype to set color of
const int	**color**	Input	Color for types new object creation

UF_OBJ_set_cre_color_to_def (view source)

Defined in: uf_obj.h

Overview
Make the work part's type/subtype color setting use the default color setting.

Environment
Internal and External

See Also
For information on type, subtype and properties click here.

Required License(s)
gateway

```
int UF_OBJ_set_cre_color_to_def
(
    const int type,
    const int subtype,
    const int property
)
```

const int	**type**	Input	Type to set color of
const int	**subtype**	Input	Subtype to set color of
const int	**property**	Input	Property of type/subtype to set color of

UF_OBJ_set_cre_line_font (view source)

Defined in: uf_obj.h

Overview
Set the specified type and subtype's line font object creation setting in the work part. If the type is not a valid object type or if no object creation settings exist for the type and subtype combination then an error is returned.

Environment
Internal and External

See Also
See the line_font values in the types and symbols
For information on type, subtype and properties click here.

Required License(s)
gateway

```
int UF_OBJ_set_cre_line_font
(
    const int type,
    const int subtype,
    const int property,
    const int line_font
)
```

const int	**type**	Input	Type to set line font of
const int	**subtype**	Input	Subtype to set line font of
const int	**property**	Input	Property of type/subtype to set line font of
const int	**line_font**	Input	Type's new object creation line font

---

UF_OBJ_set_cre_line_font_to_def (view source)

Defined in: uf_obj.h

Overview

Make the work part's type/subtype line font setting use the default line font setting.

Environment

Internal and External

See Also

For information on type, subtype and properties click here.

Required License(s)

gateway

```
int UF_OBJ_set_cre_line_font_to_def
(
    const int type,
    const int subtype,
    const int property
)
```

const int	**type**	Input	Type to set line font of
const int	**subtype**	Input	Subtype to set line font of
const int	**property**	Input	Property of type/subtype to set line font of

---

UF_OBJ_set_cre_width (view source)

Defined in: uf_obj.h

Overview

Set the specified type and subtype's width object creation setting in the work part. If the type is not a valid object type or if no object creation settings exist for the type and subtype combination then an error is returned.

Environment
Internal and External

See Also
See the width constants in the types and symbols
For information on type, subtype and properties click here.

Required License(s)
gateway

```
int UF_OBJ_set_cre_width
(
    const int type,
    const int subtype,
    const int property,
    const int width
)
```

const int	**type**	Input	Type to set width of
const int	**subtype**	Input	Subtype to set width of
const int	**property**	Input	Property of type/subtype to set width of
const int	**width**	Input	Type's new object creation width

UF_OBJ_set_cre_width_to_def (view source)

Defined in: uf_obj.h

Overview
Make the work part's type/subtype width setting use the default width setting.

Environment
Internal and External

See Also
For information on type, subtype and properties click here.

Required License(s)
gateway

```
int UF_OBJ_set_cre_width_to_def
(
    const int type,
    const int subtype,
    const int property
)
```

const int	**type**	Input	Type to set width of
const int	**subtype**	Input	Subtype to set width of
const int	**property**	Input	Property of type/subtype to set width of

---

UF_OBJ_set_def_cre_color (view source)

Defined in: uf_obj.h

Overview
Sets the default color object creation setting. Does not work for sheet bodies or solid bodies. Use UF_OBJ_set_cre_color_to_def or UF_OBJ_set_cre_color for these specific object types.

Environment
Internal and External

See Also
See the color string constants in the types and symbols

Required License(s)
gateway

```
int UF_OBJ_set_def_cre_color
(
    const int color
)
```

const int	**color**	Input	New default object creation color

UF_OBJ_set_def_cre_line_font (view source)

Defined in: uf_obj.h

Overview
Sets the default line_font object creation setting.

Environment
Internal and External

See Also
See the line_font in the types and symbols

Required License(s)
gateway

```
int UF_OBJ_set_def_cre_line_font
(
```

const int line_font
)

const int	**line_font**	Input	New default object creation line_font

---

UF_OBJ_set_def_cre_width (view source)

Defined in: uf_obj.h

Overview
Sets the default width object creation setting.

Environment
Internal and External

See Also
See the width constants in the types and symbols

Required License(s)
gateway

int UF_OBJ_set_def_cre_width
(
const int width
)

const int	**width**	Input	New default object creation width

UF_OBJ_set_face_analysis (view source)

Defined in: uf_obj.h

Overview
Sets the face analysis attribute of a face. If the attribute is TRUE, the face is displayed using analysis data in face analysis views. The parameters defining the details of the display are specified by UF_DISP_set_srfanl_params.

Environment
Internal

See Also
UF_DISP_set_srfanl_params

Required License(s)
gateway

int UF_OBJ_set_face_analysis
(

```
    tag_t face,
    logical srfanl
)
```

tag_t	**face**	Input	Object identifier of the face
logical	**srfanl**	Input	Shaded status of the face TRUE = Display with analysis data in face analysis views. FALSE = Display without analysis data

---

UF_OBJ_set_font (view source)

Defined in: uf_obj.h

Overview
Modifies the font of an object. Use UF_OBJ_ask_display_properties to determine the object's current font.

Environment
Internal and External

See Also
UF_OBJ_ask_display_properties
See the string defined font constants in in the types and symbols

Required License(s)
gateway

```
int UF_OBJ_set_font
(
    tag_t object_id,
    int font
)
```

tag_t	**object_id**	Input	Object identifier of object whose font is to be changed.
int	**font**	Input	font to assign.

UF_OBJ_set_layer (view source)

Defined in: uf_obj.h

Overview
Modifies the layer of an object. Use UF_OBJ_ask_display_properties to determine the object's current layer.

If you attempt to change the layer of an object which is on a reserved layer (greater than UF_LAYER_MAX_LAYER, or zero), the request will be ignored.

If you attempt to change the layer of an object which is on a user-accessible layer (UF_LAYER_MIN_LAYER <= layer <= UF_LAYER_MAX_LAYER) to a reserved layer, the request will be ignored, except that objects of type RM_view_type and RM_cs2_vertex_type may be moved to layer 0, which is where these object types belong, so that parts where these object types were erroneously moved off of layer 0 may be repaired.

Environment
Internal and External

See Also
UF_OBJ_set_layer_allow_hidden
UF_OBJ_ask_display_properties

History
In V18.0 attempts to move change the layer of objects on reserved layers will be ignored.

Required License(s)
gateway

int UF_OBJ_set_layer
(
tag_t object_id,
int layer
)

tag_t	**object_id**	Input	Object identifier of object whose layer is to be changed.
int	**layer**	Input	layer to assign.

UF_OBJ_set_layer_allow_hidden (view source)

Defined in: uf_obj.h

Overview
Modifies the layer of an object. Use UF_OBJ_ask_display_properties to determine the object's current layer.

This function allows movement to and from the hidden layers.

Environment
Internal and External

See Also
UF_OBJ_set_layer
UF_OBJ_ask_display_properties

History
Original release was in NX 3.0.3

Required License(s)
gateway

int UF_OBJ_set_layer_allow_hidden
(
tag_t object_id,
int layer
)

tag_t	**object_id**	Input	Object identifier of object whose layer is to be changed.
int	**layer**	Input	layer to assign.

UF_OBJ_set_line_width (view source)

Defined in: uf_obj.h

Overview
Modifies the line_width of an object. Use UF_OBJ_ask_display_properties to determine the object's current line width.

Environment
Internal and External

See Also
UF_OBJ_ask_display_properties
See the string defined line-width constants in the types and symbols .

Required License(s)
gateway

int UF_OBJ_set_line_width
(
tag_t object_id,
int line_width
)

tag_t	**object_id**	Input	Object identifier of object whose line_width is to be changed.
int	**line_width**	Input	line_width to assign.

UF_OBJ_set_name (view source)

Defined in: uf_obj.h

Overview
Sets the name of a object. You can only set the name of an object which has an object type defined in the uf_object_types.h header file. Note that the name set for an object of type UF_occ_instance_type is not the instance name; the instance name can be set using UF_ASSEM_rename_instance.

Environment

Internal and External

Required License(s)
gateway

int UF_OBJ_set_name
(
tag_t object_id,
const char * name
)

tag_t	**object_id**	Input	Object identifier of object whose name is to be set.
const char *	**name**	Input	Name to assign to object. The length of the name is limited to UF_OBJ_NAME_LEN characters.

UF_OBJ_set_name_origin (view source)

Defined in: uf_obj.h

Overview
Sets the origin of an object name. This origin is only used as the location to display the name when name display is enabled with UF_DISP_set_name_display_status. This routine will return an error if the object is not named.

Environment
Internal and External

See Also
UF_DISP_set_name_display_status
UF_OBJ_ask_name_origin

History
Originally released in V17.0

Required License(s)
gateway

int UF_OBJ_set_name_origin
(
tag_t object_id,
double origin [3]
)

tag_t	**object_id**	Input	Object identifier of object whose name origin is to be set.
double	**origin [3]**	Input	Origin in absolute coordinates where the name is to be displayed

---

UF_OBJ_set_partially_shaded (view source)

Defined in: uf_obj.h

Overview
Sets the partial shading attribute for a solid face or for all faces in a solid body.

Environment
Internal and External

Required License(s)
gateway

```
int UF_OBJ_set_partially_shaded
(
    tag_t object,
    logical shaded
)
```

tag_t	**object**	Input	Object identifier of solid face or body to set attributes for
logical	**shaded**	Input	Shaded status of the solid face TRUE = set shading FALSE = unset shading

---

UF_OBJ_set_translucency (view source)

Defined in: uf_obj.h

Overview
Sets the translucency of faces, facets and solid/sheet bodies.

Environment
Internal and External

History
Introduced in NX 2.0.2

Required License(s)
gateway

```
int UF_OBJ_set_translucency
(
    tag_t object,
    UF_OBJ_translucency_t translucency
)
```

tag_t	**object**	Input	The object for which you are setting the translucency It must be of one of the following types: UF_solid_type, with one of these subtypes: UF_solid_body_type UF_solid_face_type

			UF_faceted_model_type UF_component_type
UF_OBJ_translucency_t	**translucency**	Input	percent translucent 0 - 100